PERFORMANCE AND POWER MANAGEMENT FOR MULTI-CORE PROCESSORS

A Dissertation
Presented to
The Academic Faculty

By

Xinwei Chen

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2018

PERFORMANCE AND POWER MANAGEMENT FOR MULTI-CORE PROCESSORS

Approved by:

Dr. Sudhakar Yalamanchili, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Yorai Wardi, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Saibal Mukhopadhyay
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Linda M Wills
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ada Gavrilovska
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 4, 2018

To my husband Dr. Emory Hsu and my parents.

ACKNOWLEDGEMENTS

This thesis would not be possible without the concerted effort of many people. Foremost are my advisers Dr. Sudhakar Yalamanchili and Dr. Yorai Wardi. I have been fortunate enough to have gotten to know them over the past few years. Through both of their hard work and mentorship, I have learned not only technical aspects, but also enjoyed getting to understand the process of research. Hopefully I have absorbed some of their brilliance in computer science. I would also like to thank Dr. Magnus Egerstedt and Dr. Saibal Mukhopadhyay for their feedback and insight as my reading committee members, and Dr. Linda Wills and Dr. Ada Gavrilovska for their input on my thesis and during the defense.

I would also like to thank Dr. Leonardo Piga and Dr. Indrani Paul for their mentorship during my internship at AMD Corporation in Austin, Texas. This real-world internship outside of Georgia Tech provided a great learning experience in seeing computer science from a different approach.

Of course, daily work in the CASL (Computer Architecture and Systems Laboratory) lab means getting to know many wonderful colleagues. These include Jeffrey Young, Chad Kersey, Will Song, Jin Wang, Haicheng Wu, Eric Anger, Si Li, Minhaj Hassan, He Xiao, Karthik Rao, Blaise Tine, and Bahar Asgari. GRITS lab colleagues include Matt Hale, Tina Setter, Sebastian Ruf, Li Wang, Mara Santos, Daniel Pickem, Yancy Diaz-Mercado, Zak Costello, and Usman Ali. Other GT colleagues include Jen-cheng Huang, Lifeng Nai, Alexis Champsaur, Hongteng Xu, Yu Liu, Alexander Merritt and Vinson Young. Through the successes and the challenges, I am so glad to have the good fortune to be able pick your brains and share in both the exhilaration and the tedium of computer science research. Your teamwork and friendship are so invaluable to me, and I am truly indebted to you all.

There are many, many other individuals who contributed as well, and to each and every person who I have encountered, I owe my thanks and deep appreciation.

Finally, I am in gratitude to my friends and family. Your encouragement and support form the bedrock of my daily life, and this thesis would not be possible without you all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This dissertation addresses the problem of power and performance management for various computing systems, from single voltage island multicore processors to power-constrained extreme scale cloud systems. Balancing power and performance in modern computing systems is a complex optimization problem. This challenge is addressed by the statement of this thesis: Improving performance and power consumption in modern computing systems will require new techniques, and the body of control theories can provide the basis for such solutions. This thesis addresses this problem through three main contributions:

- Effective and efficient power & performance management techniques in a single voltage island multi-core processor.
- Maximizing power efficiency under a power cap in a multi-core processor that is composed of several voltage islands.
- A hierarchical power management technique to improve performance and energy efficiency under power budgets in a cloud system.

The first topic is comprised of 1) throughput regulation, 2) power regulation, and 3) power efficiency optimization for single voltage island multicore processors. A throughput-frequency model is obtained by IPA analysis, while a power-frequency model is obtained by a system identification approach. These models are generic and can be applied to various applications. They provide a foundation for the on-line optimization of power efficiency in multi-core processors.

The second topic addresses the problem of optimizing power efficiency in a many-core processor under power caps, such as those found in servers in the nodes of cloud systems. Given a power budget, we provide two techniques for improving the power efficiency: 1) an on-line optimization technique for maximizing throughput, 2) a dynamic power regula-

tion technique that dynamically distributes power across the processor based on workload variation, which is an extension of the power regulation technique in the first topic.

Finally the third topic addresses the problem of performance and energy efficiency improvement for cloud systems under power budgets. This work presents a hierarchical power gating & power shifting (HPGPS) technique for bulk synchronous parallel applications in cloud computing systems. Nodes that are otherwise waiting to be synchronized are power gated and their power budgets are redistributed to other high workload nodes, thus reducing the penalty of workload imbalances across the system. This hierarchical power management scheme is scalable to extreme scale cloud computing systems.

By examining these topics, this thesis contributes to improving the power consumption and performance of computing systems from single processor architectures to full scale cloud systems.

CHAPTER 1
INTRODUCTION

Power efficiency is a major concern in all components of computing systems, from mobile devices to servers to data centers. This concern is exacerbated as modern applications process a growing volume of data which requires increasing performance and energy. Data centers consume approximately 2% of all electricity use in the U.S. [1], and concerns about the impact of energy consumption in these facilities continue to grow. This imposes a challenge for balancing performance and power consumption in modern computing systems. In general, performance and power consumption are opposing metrics, where improving one is often achieved at the expense of the other. Heuristic solutions may no longer satisfy the demand for systematic power and performance management. Formal techniques with theoretical bases that are robust, stable, and efficient are needed. This challenge is addressed by the following thesis statement: Improving performance and power consumption in modern computing systems will require new techniques, and the body of control theories can provide the basis for such solutions.

Optimization of power consumption in these computing systems is a multi-step process. For example, a data center is composed of thousands of nodes, and each node is composed of several processors. A processor is composed of one or more voltage islands, and each voltage island is composed of one or more cores. The process of performance and power usage at each component is different based on their architecture characteristics; thus, power efficiency optimization must be individually tailored to each component. This thesis presents power and performance management for cores, processors, and cloud systems, with the goal of creating optimized components and laying the foundation for further synergies in power efficiency of computing systems.

To that end, this dissertation presents formal control approaches for balancing power

and performance in order to achieve better performance at lower power expense. The work is categorized into three research themes: 1) power regulation, throughput regulation, and power efficiency optimization in a single voltage island in a processor; 2) power efficiency optimization across multiple voltage islands in a multi-core processor under power caps; and 3) performance and energy efficiency improvement in a cloud computing system that is composed of thousands or millions of voltage islands. Figure 1.1 shows the relationship between contributions of this work.

This thesis developed dynamic models for throughput and power that adjust well to workload variations. Those models are general and can be applied to various kinds of computing frameworks. Based on those models, we use feedback controllers for throughput regulation and power regulation. The controllers are based on integrators for variable gain designed for stabilizing the closed-loop system as well as for rapidly responding to changing workload in short time frames. The feedback control is robust with respect to model uncertainties and computing errors in the loop, and they exhibit fast convergence despite such errors.

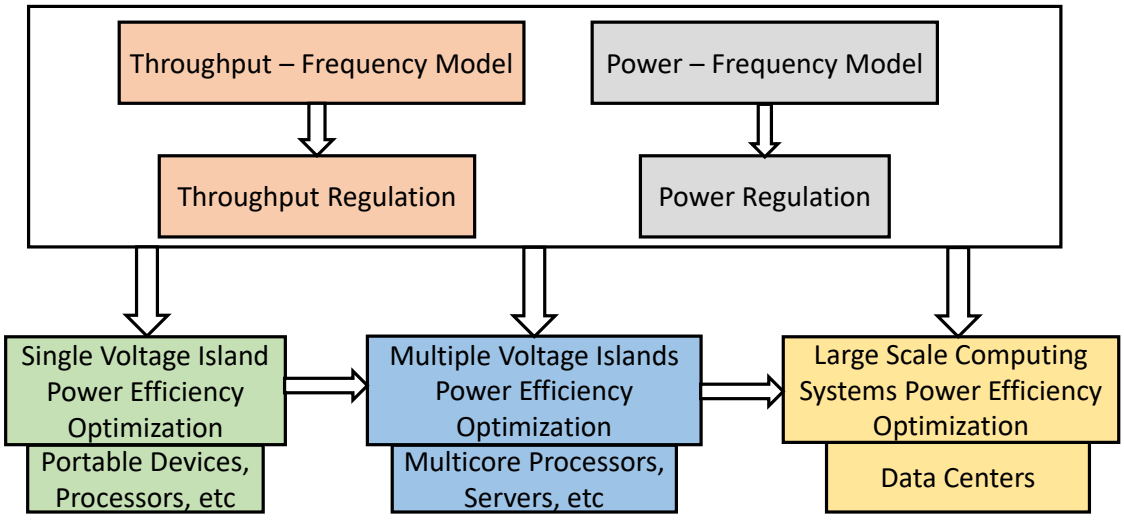

Figure 1.1: The Structure of Contributions

The next section describes the main contributions of this work, followed by a summary

of the structure of this thesis. The main contributions have also been presented in several publications [2][3][4][5][6].

1.1 Contributions

This thesis makes the following key contributions

The first contribution of this work is effective and efficient power and performance management in a single voltage island in a processor. This single voltage island may contain one or more cores. We created three designs for performance and power management. The first design is an adaptive gain throughput regulator that adjusts operating frequencies. This regulator maintains fixed throughput in the presence of dynamically varying parallelism and inter-instruction dependencies in the instruction stream. The second design is an adaptive gain power regulator that can control the power of cores residing in a single voltage island to desired set points under a variety of program workloads. Finally, the third design is an on-line power efficiency optimization controller based on stochastic approximation approaches that balance the power consumption with throughput.

The second contribution of this work is to maximize of power efficiency in a power capped processor that is composed of several voltage islands. Cores residing in the same voltage island operate at the same frequency, while different voltage islands can operate at different voltage and frequency levels. Based on the understanding of frequency-power and frequency-throughput relationship that is developed in the first contribution, we extend the single voltage island optimization to multiple voltage islands that are composed of many cores. Our approach modulates power to performance variation across multiple voltage islands by dynamically assigning the frequency and voltage level for each voltage island so that the overall power efficiency of the processor is maximized. The clock frequency presents trade-offs between performance and power. Hence, two intuitive methods to improve power efficiency are 1) reducing power under fixed throughput, and 2) increasing throughput under power caps. This work is based on the second approach. We present

two designs: optimization and dynamic regulation. The optimization design assigns the frequency and voltage levels of voltage islands distributed in the processor in order to maximize performance under power caps. On the other hand, the regulation design is based on the on-line power tracking technique in the first contribution. We dynamically distribute power targets for voltage islands depending on the underlying application characteristics.

The third contribution of this work is a hierarchical power management approach that can improve performance and energy efficiency under a power budget in a cloud computing system. A cloud computing system is composed of thousands to perhaps millions of nodes, and each node is composed of multiple processors. Performance variation is a significant problem for efficient power management of High Performance Computing (HPC) applications in large scale cloud systems. Among the most frequently used HPC applications in modern cloud systems are Bulk Synchronous Parallel (BSP) applications. A BSP is composed of parallel computations on each node, communication among nodes, and barrier synchronizations. The application behaviors vary significantly across nodes. The nodes that arrive at the barrier first must spend idle time waiting for other nodes to arrive at the barrier. The performance is limited by the slowest node since the other nodes have to wait on barrier synchronization. This idle waiting consumes power but produces no effective throughput - thus it is a major source of inefficiency. The key idea of this work is to power-gate the nodes that have finished computation arriving at the barrier, and shift the saved power from those power-gated nodes to other nodes that are still under computation so that those computation nodes are sped up while staying under the system power budgets. Consequently, the program completion time is reduced under the preassigned power budget. The hierarchical power-shifting & power-gating approach is scalable across system sizes.

Thus, this dissertation contributes to improving the power consumption and performance of computing systems from single processor architectures to full scale cloud systems.

1.2 Organization

In brief, Chapter 2 provides an overview of power and performance management in computing systems. Chapter 3 describes the evaluation frameworks for experiments in this work. The first contribution is presented in Chapter 4, Chapter 5, and Chapter 6. The second contribution is presented in Chapter 7. The third contribution is presented in Chapter 8. A more detailed overview of the chapters is below.

Chapter 2 describes the landscape for power and performance management in computing systems. It contains an overview of techniques used to regulate performance and power, as well as improve power efficiency for multi-core processors. In addition, this chapter includes a description of energy efficiency improvement techniques for cloud computing systems as they scale, and the major existing methods for optimizing the performance and energy efficiency of bulk-synchronous parallel applications.

Chapter 3 presents the evaluation frameworks for experiments conducted in this work, including a cycle level architecture simulator and an Intel Haswell processor. This chapter also introduces benchmarks that are used for testing.

Chapter 4 presents the method for regulating processor throughput under various workloads. An on-line sensitivity analysis technique is developed to model the relationship between throughput and operating frequencies. A variable gain feedback controller is developed for regulating the throughput of multi-core processors. Implementation in an Intel Haswell 4-core processor demonstrates less than 2.8% throughput tracking errors.

Chapter 5 demonstrates the method for regulating power of multi-core processors. A system identification model is developed to estimate the power-frequency relationship. An adaptive gain feedback controller for power regulation in multi-core processors is presented. Implementation in an Intel Haswell 4-core processor shows less than 5.7% power tracking errors.

Chapter 6 presents the method for improving power efficiency of cores as well as pro-

cessors. Based on the performance model developed in Chapter 4 and the power model developed in Chapter 5, we further explore the trade-off between power and performance by on-line optimization. Compared to Linux Conservative governor, our approach improves the power efficiency (measured as Throughout-per-Watt) up to 15.91%.

Chapter 7 presents two techniques for improving performance and power efficiency for power capped processors that are composed of multiple voltage islands. Those two techniques are: an on-line optimization controller for optimizing performance in a power capped processor, and a dynamic power regulator which is an extension of the power regulation presented in Chapter 5 to leverage power for performance across multiple voltage islands.

Chapter 8 presents the method for improving energy efficiency and performance of cloud systems executing bulk-synchronous applications. This hierarchical power gating and power shifting (HPGPS) technique is scalable to extreme scale cloud systems and can tolerate large network latency. Experiments in an AMD in-house simulator show that HPGPS can achieve up to 1.5% energy saving.

Finally, Chapter 9 summarizes the contributions and conclusions from this thesis as well as contemplates future areas of research.

CHAPTER 2
LITERATURE REVIEW

2.1 Power Regulation For Multi-core Processors

Various techniques from the field of system and control have been used to regulate the power consumption in multi-core processors. Several power regulation algorithms utilize open-loop optimization strategies under the assumption that power consumption of a processor at each supply voltage level can be estimated accurately [7] [8]. Those methods can work effectively when the system is running programs that have similar patterns as the ones used for empirical analysis. However they may present severe performance degradation or even power constraint violation when workloads vary significantly.

Feedback control is an effective way to regulate power in multi-core processors because of its theoretically guaranteed accuracy and robustness [9] [10]. The parameters in the control model can be determined by off-line analysis of extensive workload [11] or on-line system analysis [12]. Proportional controllers [13] and PID controllers [14] are implemented in hardware architecture design to dynamically change supply voltages adapting to power constraints in muli-core processors. Wang et al. [15] regulate per-core power under various workload by a model estimator and shifting power between CPU cores and memory components based on MPC (Model Predictive Control) theory.

2.2 Throughput Regulation For Multi-core Processors

In multi-core processors, a wide range of throughput regulation techniques using control theoretic approaches have been explored. One fuzzy flow regulation technique [16] uses fuzzy logic to intelligently control the input flow rate in the chip network according to traffic dynamism and interconnection network status. PI and PID controllers are used to

balance resource utilization and immigrate tasks in multi-core processors [17]. The work by Brinkschulte et al. [18] regulate the IPC (instruction per second) rate by switching CPU resources among threads using a proportional controller. Almoosa et al. [19] presented an on-line throughput regulation method using IPA (Infinitesimal Perturbation Analysis) by modeling the instruction-sequences as stochastic DEDS (Discrete Event Dynamic Systems). However this work does not provide as robust regulation as in [6], where memory-bounded instructions are part of the instruction flow model.

2.3 Optimizing Power Efficiency For Multi-core Processors

In recent years, several classes of techniques have been developed to improve power and energy efficiency in multicore processors. Two such classes are resource allocation and DVFS [20]. Resource allocation techniques dynamically re-assign computing resources according to workload variations. Example techniques include virtual machine scheduling [21], task migration [22] and thread scheduling [23]. Those techniques usually depend on the ability to predict or detect application phases. In reference [24] an approach is described for predicting power load variations using performance counter information and controlling the power module configuration accordingly. Ref. [25] maximized performance while maintaining power and thermal constraints by a runtime optimization policy, which is based on the training of power and performance statistics from simulations across a group of benchmarks. Ref. [26] predicted the system performance state from readily available input features, such as the occupancy state of a global service queue, using Supervised Learning technique, and then used this predicted state to look up the optimal power management action, e.g. voltage frequency setting, from a pre-computed policy table.

There has been a surge of power efficiency optimization techniques [27] [28] [29] applying DVFS (Dynamic Voltage and Frequency Scaling) to multi-core processors, where operating frequencies and voltages are reduced to diminish power consumption without performance loss. In order to exploit power for performance, various scheduling tech-

niques to adapt frequency and voltage levels according to workload variations and resource utilization are developed. These techniques decrease CPU frequency and voltage during 1) memory-intensive phases in applications, and 2) internal communication phases in parallel programs [30]. These scheduling technologies can be implemented in different fashions, from low-level hardware architecture to high-level runtime optimization policies.

Low-level DVFS implementations include hardware architecture design, and operating system level power management strategies [31] [32]. The work in [33] detects L2 cache misses and instruction-level parallelism in hardware to leverage the low-usage period of CPU. However, the implementation for architecture level power management schemes requires hardware support, which is complicated and sometimes impractical. In OS-system level, CPU frequencies and voltages can be set in response to runtime application behavior prediction, which is based on resource utilization information provided by OS kernels [34].

To take full advantage of DVFS techniques effectively and efficiently, high-level implementations including power and performance modeling and control theoretic approaches are used. In power and performance models, potential benefits or penalties of different frequency and voltage states can be predicted before actual occurrences [35]. Some models are established according to detailed analysis of certain architecture [36], while others are linear models for power estimation based on CPU utilizations [10][37]. Meng et al. [25] design an application based optimization policy by maximizing performance under power and thermal constraints. They study application runtime characteristics such as network traffic, workload, and memory intensive patterns, and use them to construct an off-line model to determine the optimized Voltage-Frequency setting. Recent work has developed simple and real-time power models with low implementation overhead based on performance counters and OS utilization metrics [11] [38]. Srikantaiah et al [39] measure disk, network, and CPU utilization, paving the way for modeling consolidated power and performance so as to minimize power consumption.

Formal control theoretic approaches have been used to optimize power efficiency for

multicore processors using DVFS [40][41]. They are broadly classified as optimal control, especially Model Predictive Control (MPC) [42] and formal feedback control [43]. The mathematical model developed by [44] based on Model Predictive Control (MPC) describes the workload variation in multicore environment and changed operating frequency and voltage accordingly to save energy. [45] used MPC techniques to minimize multicore processor system energy under temperature constraints. The researchers in [15] regulated power under temperature constraints by integrating dynamic cache size to shift power among cores via piecewise linear model.

Closed loop feedback controllers are also used to manage power and performance in multicore processors. Proportional controllers [13] and PID controllers [14] are implemented in hardware architecture design to dynamically change supply voltages adapting to power constraints. The work in [43] uses a PID controller with the synchronizing queue occupancy as the input in multi-clock domain processors. [12] used an online feedback controller to regulate the power consumption for a multicore processor with theoretically proved robustness and stability.

2.4 Optimizing Energy Efficiency Under Power Budgets in Data Center Systems

In large scale computing systems, performance is limited by the available power [46]. A power budget may be imposed by the existing power provisioning facilities as well as high power consumption issues. Two well-known approaches, DVFS[47][48], and power-shifting and power-gating [49], have been developed to increase power efficiency under power caps in data centers, leading to better performance.

DVFS is a widely used technology that allows the CPU clock frequency and supply voltage to be changed dynamically [50]. DVFS trades processor performance for lower power consumption in cluster nodes. Lower frequencies and voltages lead to lower power, making power-up active computing nodes possible. As a result, the execution time for nodes in critical paths is reduced. The overall performance measured in BSP (Bulk Syn-

chronous Parallelism) programs improves because of shorter synchronization time. In addition, a node frequency and power can be switched to the lower power status during phases of communication in parallel programs [30].

A runtime mechanism is presented by [51] for slack prediction and slowing down of critical path computation for the benefit of energy saving. However, their execution model consisting of multiple steps is assumed to contain a compute followed by communication globally at each step in BSP. This approach does not provide as good energy efficiency for applications with time-dependent processing patterns as [52], where an Energy Template is proposed to identify idle states of the processor cores. This template information is passed to MPI (Message Passing Interface) runtime to achieve potential energy saving. A job scheduling policy is proposed by [53] to allocate both processor and power resources to all jobs at the same time. It distributes the available power among the jobs, assigning optimal CPU frequency to each of the selected jobs. Several software-controlled dynamic power management algorithms have been explored [54] [55] [36] [56] by using DVFS to expand computation into slack that occurs during communication for synchronization, thus reducing energy consumption. The work by [57] tracks the idle time spent by a processor waiting for other processors to reach the barrier in the program and reduces the frequency of the processor in order to reduce or eliminate these idle time. Energy aware optimization methods have been applied to MPI by identifying communication phases of parallel applications and using DVFS during such phases to conserve power [58] [30] [59] [60].

Power-gating is a technique that shuts off the power supply of a logic block by inserting a gate or sleep transistor [61]. There is virtually no power consumption in the gated block. The supply voltage is significant lower than what is used in standard DVFS, contributing to aggressive reduction in power consumption. Under a strict power budget, the power saved from gated nodes can be used by other nodes in the data center system, which is called power-shifting. Thus, power-gating and power-shifting make full use of computing resources. However, power-gating should not be used liberally because there is usually a

performance overhead associated with entering and exiting the power gated states.

Power-gating and power-shifting techniques have emerged as energy efficiency solutions for data centers. To complement existing DVFS techniques, per-core power-gating (PCPG) [62] has been developed to reduce leakage power when computing resource utilization exhibits high variability. The Booster [63] runtime system re-balances parallel workloads by shifting power among function units within a processor. The work by [64] provides a power-gating strategy which makes power saving possible without performance loss. The transition of power states from power-gated mode to full power mode is smooth and vice versa. Moreover, Lefurgy et al. [58] are among the first to explore the power-shifting technique in node level under power caps. Several workload-guided policies for dynamically allocating power under static budgets among node servers have been developed in the past few years [49]. Furthermore, power-gating and power-shifting techniques have been extended system wide, resulting in several energy efficiency optimization approaches, including a just-in-time network power-shifting method [65] that enables power-shifting from nodes to nodes. Piga et al.[66] developed a Gate & Shifting method that allows one to mitigate the BSP communication imbalance among nodes. They power-gate waiting processes and shift the remaining power budget to the processes that are in the critical execution paths. However, this power-shifting approach requires communication across the data center system, resulting in significant communication delay. Therefore, it may not be applied to extra-scale data center systems.

CHAPTER 3
EXPERIMENTAL FRAMEWORKS

This chapter presents the experimental frameworks used in this thesis, including a full system cycle level architecture simulator, and an Intel Haswell 4-core processor. A brief introduction of 3D architectures is also provided.

3.1 System Architecture of a 3D Multi-core Processor

This work considers a 3D x86 multi-core processor architecture that is composed of a homogeneous 16 core die, a last level cache (LLC) die and a dynamic random-access memory (DRAM) die stack together as shown in Figure 3.1. The bottom die of the stack is the cores modeled at 16nm technology. The cores reflect a typical out-of-order core design with 5 typical partitions (FE: pipeline frontend and L1 instruction cache, SCH: out-of-order scheduler, DL1: data L1 cache, INT & FPU: integer and floating point unit). The core die is divided into 4 voltage islands, with 4 cores on each voltage island.

On top of the processor die is LLC stack. In this paper we study the core and cache, not including the next level memory hierarchy. The cache hierarchy includes a 16 KB private L1 data cache and a shared L2 LLC cache residing on the next tier with 16 banks (2MB each). The hit time for L1 cache is one cycle. The cache coherent protocol is directory-based MESI co-located in the LLC cache. Memory controllers are integrated in the DRAM stack. The interconnection network has 128-bit channels comprised of interfaces and routers. The routers are connected in a 2D torus, and the network interfaces connect to the L2 cache banks and memory controllers. On top of the LLC tier is the DRAM stack, where DRAM layers are stacked. The DRAM die is divided into 16 vertical vaults, with each vault having one memory controller connected to the DRAM partitions by a data channel.

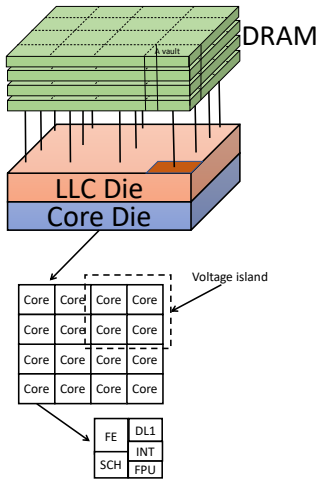

Figure 3.1: 3D Architecture Overview

3.2 A Full System Cycle Level Computer Architecture Simulator: Manifold

In this section, we describe a 3D modern architecture simulator, Manifold [67]. Manifold enables cycle-level full system processor simulation, i.e. application and operating system binaries driving cycle-level models of cores, coherent caches, on-chip networks, and the DRAM system. Manifold also supports dynamic voltage frequency scaling and is coupled to energy and thermal models via the Energy Introspector multi-physics modeling library [68], as shown in the flow chart in Figure 3.2. Figure 3.3 shows the Manifold platform system model. The pipeline execution and cache activity during execution are recorded by performance counters. Information collected by those counters are sent to the power libraries (i.e. McPAT [69]) to generate power traces with preset power models and technology specifications. The thermal distribution across the chip stacks is computed by the thermal library (i.e. 3D-ICE [70]). Temperature-leakage feedback is used to update the leakage power of each architecture component with its temperature level. Finally, the system monitor collects the processor performance and multi-physics information, and adaptively manipulates the operating knobs (i.e. voltage, frequency) according to our control

algorithms at runtime.

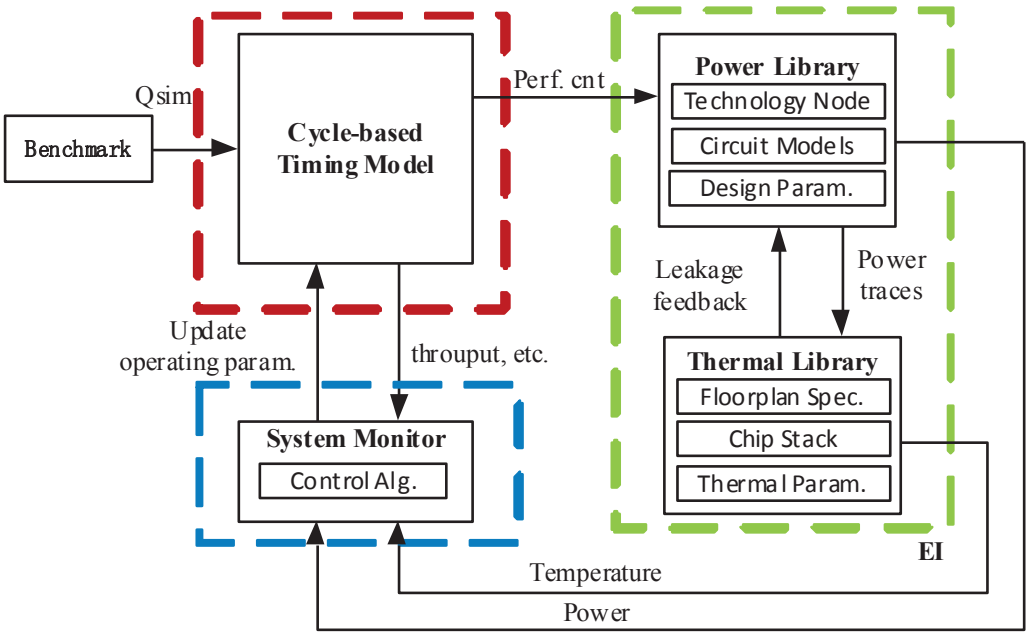

Figure 3.2: Flow Chart of Manifold Simulation

Table 3.1: System Configuration

Parameters	Out-of-order Core
Architectural Configuration	
ISA	x86 IA32
Pipeline Depth	10 stages
Fetch/Decode	4 instructions
Execution	6 Issue ports
Issue Width	4
L1 Cache	8-way 16KB/core
L2 Cache	64-way 2MB/bank, 16 banks
Physical Configuration	
Clock Frequency	0.5-5.0GHz
Supply Voltage	0.5-1.2V
Feature Size	16nm

The floor plans are illustrated in Figure 3.4. The 16 out-of-order homogeneous cores are placed on the bottom die, with 5 typical partitions (FE: pipeline frontend, SCH: out-of-

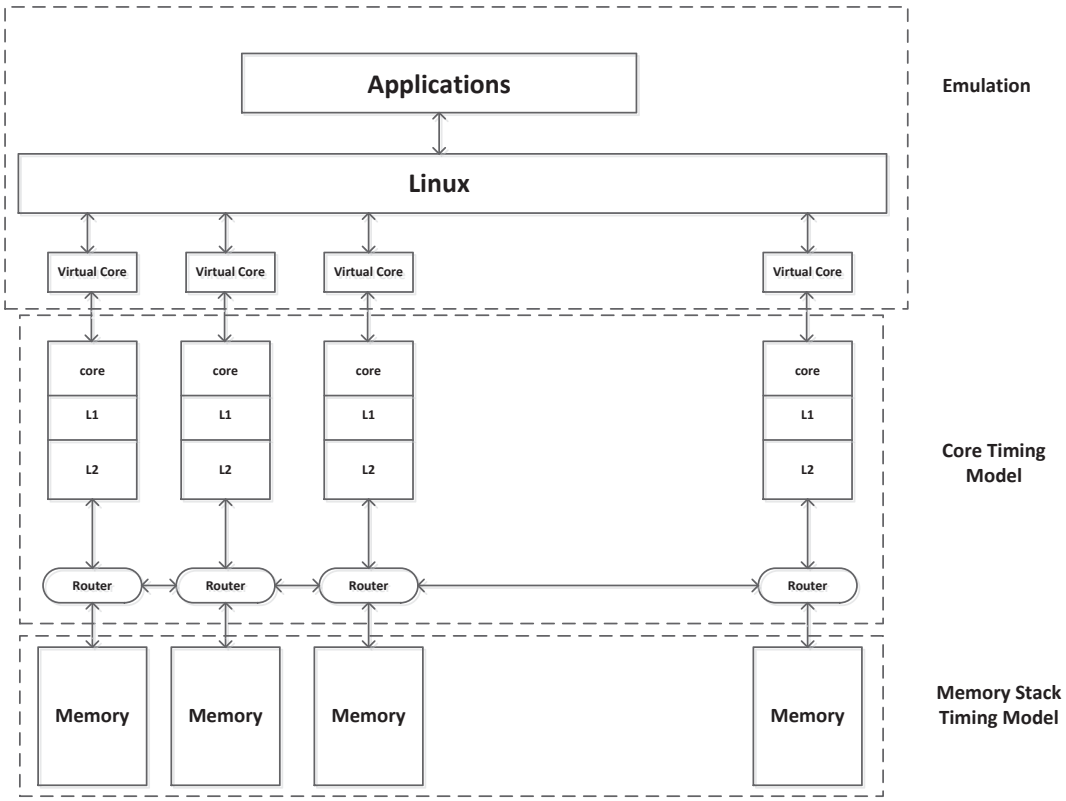

Figure 3.3: Manifold Platform Model

order scheduler, DL1: data L1 cache, INT & FPU: integer and floating point unit). These cores are interconnected by a 2D torus network. On top of the first two tiers are another 8 DRAM dies, with 16 channels, where each channel has 8 ranks (on each die) and 2 banks. The system simulation model computes the power dissipation and resulting thermal fields produced by the package. The core configuration is shown in table 3.1.

3.3 A Haswell 4-core Processor

This work also implements the designs in an Intel Haswell 4-core processor. In this section, we present the tested Intel $Core^{TM}$ i7-4770 Haswell processor that has four physical cores with a clock frequency range from $0.8GHz$ to $3.4GHz$. Since cores on the processor are residing on one voltage island, all the cores must run at the same voltage. Each core has a two eight-way 32 KB private L1 cache (separate instruction cache and data

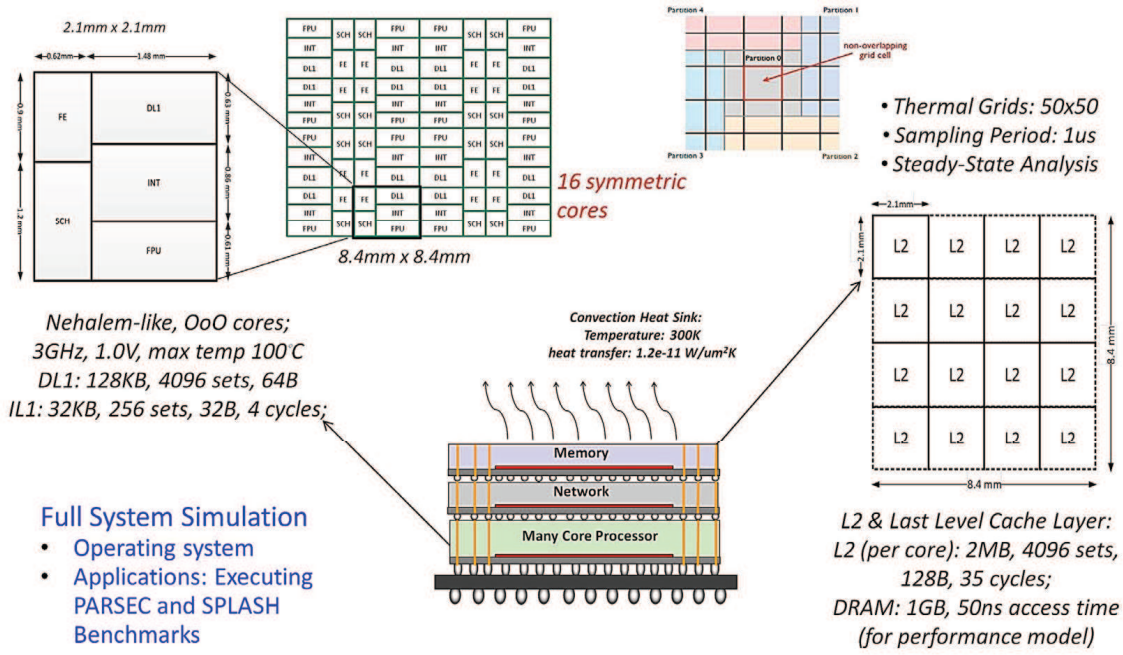

Figure 3.4: System Architecture Model

cache), a 256 KB private L2 cache (combined instruction cache and data cache), and an 8 MB shared L3 cache, with 16 GB of physical memory on board [71]. The Haswell die map is shown in Figure 3.5. Each Haswell core shares its execution resources between two threads of execution via Intel Hyperthreading [71]. We collect performance counter values using the PAPI [72] tool. PAPI allows for transparent power and energy readings via the Intel RAPL (Running Average Power Limit) interface. The throughput is measured by dividing the total number of instructions processed during one control cycle by the control cycle duration. The number of instructions proceeded is read by performance counter "PAPI_TOT_INS".The energy consumed is measured from RAPL "PP0_Energy:PACKAGE0" [72]. The frequency values are periodically set to the contents of the file "/sys/devices/system/cpu/cpu%d/cpufreq/

scaling_setspeed". The operating system in the test processor is Ubuntu 16.04.

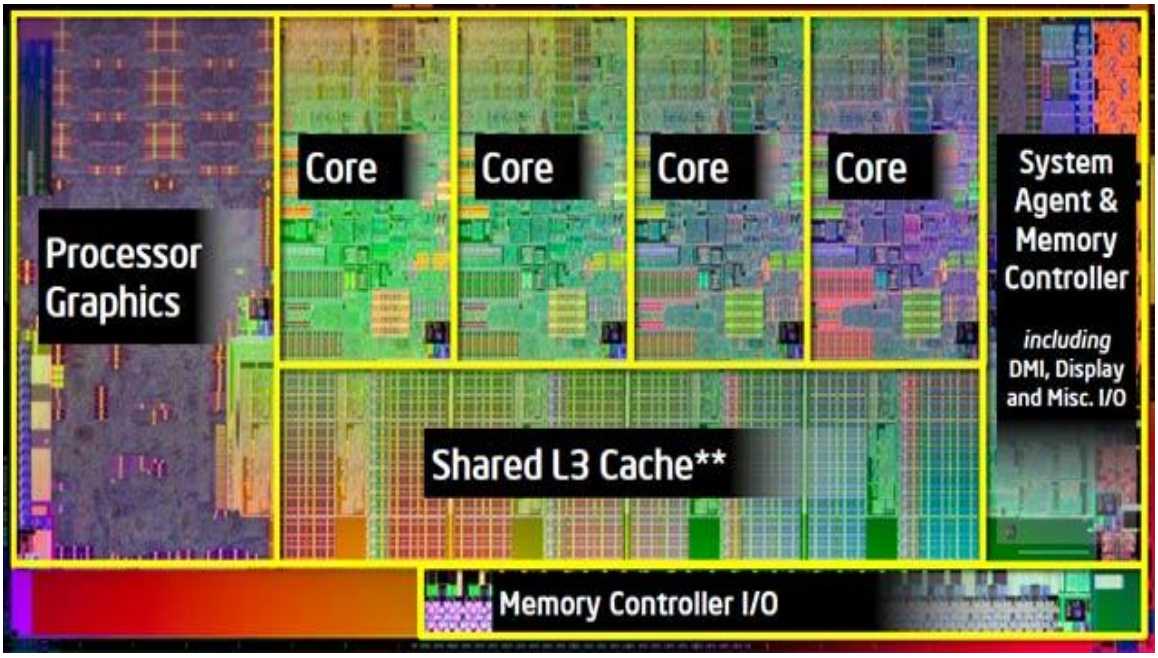

Figure 3.5: Haswell Die Map [71]

3.4 Linux Governors

The Linux operating system provides support for managing the power states of the processor through configurable software modules referred to as *governors* [73]. There are 5 CPUFreq governors in the tested Ubuntu 14.04 Linux kernel: Performance, PowerSave, Userspace, Ondemand, and Conservative [74]. The Performance governor sets the CPU statically to the maximum frequency while Powersave sets the CPU statically to the minimum frequency. The Userspace governor runs the CPU at frequencies specified by the users. The Ondemand governor aggressively makes the frequency jump to the maximum value when there is any workload and then possibly backs off when idle time increases. The Conservative governor sets the CPU frequency dynamically based on the current workload. It is like the Ondemand but differs in behavior in that the Conservative governor gradually increases or decreases the CPU speed [73]. We use the Userspace governor when implementing our designs in the Intel Haswell Processor.

3.5 Benchmarks

In this thesis, we use Splash II [75], Parsec [76], and GraphBig [77] benchmark suits. Splash II and Parsec are both from SPEC benchmark suits, which are industry standard applications used to evaluate performance and energy efficiency for computing systems [76]. GraphBig benchmarks are based on IBMs System G framework, a comprehensive set of industrial graph computing tool, cloud and solutions for Big Data used by commercial clients [78]. IBM System G can be used in many cases, such as social network analysis, anomaly detection, smarter commerce, smarter planet, cloud, telecommunication. System G includes Graph Database, Graph Visualizations, Graph Analytics Library, Graph Middleware for various hardware and distributed cluster, and Network Science Analytics tools, including: Cognitive Networks, Cognitive Analytics.

CHAPTER 4

THROUGHPUT REGULATION FOR MULTICORE PROCESSORS

This chapter addresses the problem of throughput regulation where the instruction throughput of a multi-core processor is maintained at a set target by varying core frequencies. Throughput regulation in processors presents several challenges. The first is the time-varying instruction level parallelism (ILP) exhibited by applications. Instruction and resource dependencies affect instruction flows in out-of-order cores and consequently execution time can vary significantly within an application and across different applications. Such variability is amplified in asymmetric multicore architectures comprised of cores that support varying degrees of issue width and complexity. Furthermore, communication delays between cores and other components, such as caches, DRAM, and SSDs, can rarely be predicted reliably. Threads executing on distinct cores interfere with each other in shared caches and on-chip networks introducing dynamically determined delays in instruction execution. It is therefore difficult to develop general analytic models that can relate core and chip instruction throughput to micro-architectural parameters such as frequency. All of this suggests the merit of dynamic on-line throughput regulation techniques that do not rely on static analytical models but rather continually adapt to the processors dynamics to regulate core and processor instruction throughput at set levels.

The main contributions in this chapter are:

- A variable gain controller design for regulating the throughput of modern out-of-order cores.
- A throughput model based on the on-line sensitivity analysis method that dynamically estimates the analytical relationship of throughput and core frequency.
- An evaluation of the regulator design with a full system, cycle-level multicore simu-

lator executing industry standard benchmark applications.

- Implementation and evaluation of the design in an Intel Haswell processor with various benchmarks.

4.1 A Variable Gain Controller

The purpose of the regulator described in this section is to regulate the instruction-rate of each core to a given setpoint reference. Each core is a single voltage island with its own regulator and target reference. Figure 4.1 illustrates the feedback system.

The instruction throughput of a core is measured over continuous time intervals, called *control cycles*, denoted by C_n, $n = 1, 2, 3, \ldots$, and the throughput measured during C_n is denoted by T_n. Let u_n denote the clock frequency during C_n. u_n is assumed to be assigned by the regulator at the start of C_n and maintain its value throughout that control cycle, while T_n is assumed to be measured during C_n and be obtained at the end of it. Details about the measurement of throughput will be introduced in Section 4.3 and Section 4.4. Let r be the target throughput, and the objective of the regulator is to ensure that T_n approaches r.

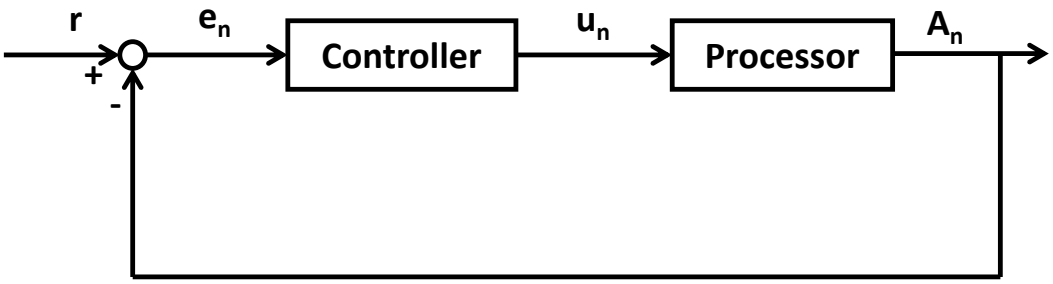

Figure 4.1: The Feedback System

The action of the controller is defined by the equation

$$u_{n+1} = u_n + A_{n+1} e_{n+1}, \qquad n = 1, 2, \ldots, \tag{4.1}$$

where A_n is its gain during C_{n+1}. The proposed design is an adaptive gain feedback regulator where A_n may change during every control cycle. e_{n+1} is the error signal for cycle C_{n+1}, which is the difference between the target and actual throughput at control cycle C_n. See below:

$$e_{n+1} = r - T_n. \tag{4.2}$$

The gain A_{n+1} is defined as [4]:

$$A_{n+1} = \xi \left(\frac{dT_n}{du_n} \right)^{-1}, \tag{4.3}$$

where $\xi \in (0,1)$ is a given constant determined experimentally to provide maximum tracking performance. The term $\frac{dT_n}{du_n}$ in Equation (4.3) is the sample derivative of the core's throughout with respect to clock frequency during C_n. We developed a queueing model using IPA (see Section 4.2) to obtain A_{n+1}. T_n can be simply computed by observing the number of instructions completed by the core during C_n and dividing it by the duration of C_n. The term $\frac{\partial T_n}{\partial u_n}$ is the sample derivative of that sample path. T_n is measured at the end of C_n, hence it can be used to compute u_{n+1} via Equation (4.1) at the start of C_{n+1}.

Next, let's take a look at the co-efficient ξ. For $\xi = 1$, the control law implements the Newton-Raphson method for solving the tracking problem. Convergence of the Newton-Raphson method is known to be robust to variations in that equation as well as to computational errors [79], and therefore we expect the control law to yield tracking regulation in the stochastic, time-varying setting under consideration. The purpose of using a factor $\xi \in (0,1)$ in Equation (4.3) is to reduce oscillations that are caused by randomness. Observations about the impact of ξ in the proposed design is presented in section 4.3.

4.2 A Throughput Model

This section describes the computation of $\frac{dT_n}{du_n}$ by using IPA (Infinitesimal Perturbation Analysis) approach. IPA is a well-known and well-tested technique for computing sample-performance derivatives (gradients) in discrete event systems and other event-driven systems with respect to controlled variables [80] [81]. Its salient feature is in simple rules for tracking the propagations associated with a gradient along the sample path of a system, by low-cost algorithms. However, this simplicity may come at the expense of statistical unbiasedness of the IPA derivatives. In situations where IPA is biased, alternative perturbation-analysis techniques have been proposed, but they may require far-larger computing efforts than the basic IPA (see [80] [81]). For the throughput regulation technique described in this work, it has been shown that IPA need not be unbiased and, as mentioned earlier, its most important requirement is low computational complexity [5].

The instruction flow through the core involves six stages : Fetch, Decode, Issue, Execute, Memory, and Commit. To quantify the throughput, we next derive the equations that describe the last four steps (see Figure (4.2)).

To start with the Issue step, consider a sequence of instructions, denoted by $I_1, I_2, \ldots$, according to their issue order. Let a_i denotes the arrival time of instruction I_i to the Reorder Buffer (ROB) in terms of clock cycles. If the instruction has a data dependency, we use $k(i)$ to denote the index of the instruction that computes the last operand required for instruction I_i . Let τ denote the core's cycle time, and denote by α_i the enqueue time of I_i, namely the time that all the operands of I_i are available and the instruction is ready to be executed. Then

$$\alpha_i \; = \; max\{ \; a_i\tau \; , \; \beta_{k(i)} \; \} + \tau. \tag{4.4}$$

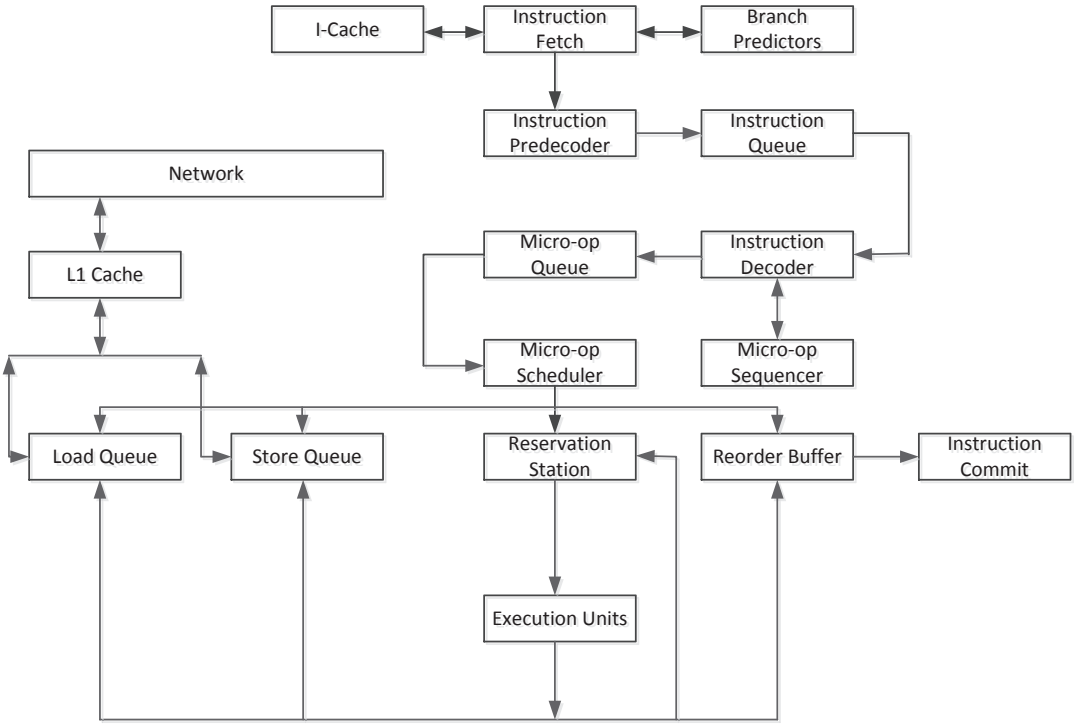

Figure 4.2: Out-of-order Execution [82]

Therefore, we have

$$\alpha_i'(\tau) = \begin{cases} \xi_i + 1, & \text{if all the operands of} \\ & \text{instruction } I_i \text{ are ready} \\ & \text{before } I_i \text{ arrives at ROB.} \\ \beta_{k(i)}'(\tau) + 1, & \text{otherwise.} \end{cases} \tag{4.5}$$

The Execute stage, assume that the execution time of a non-memory instruction I_i is approximated by $\mu_i\tau$, where μ_i is total number of clock cycles it takes the execution unit to process instruction I_i . Denote by β_i the completion time of executing I_i. Then,

$$\beta_i \;=\; \alpha_i + \mu_i\tau, \tag{4.6}$$

and we note that β_i is also the time that the result of instruction I_i becomes available as an operand for other instructions.

Thirdly, if instruction I_i is a memory instruction, the memory hierarchy is involved in the process. Let us denote the sequence of instructions I_i that are in the memory path by $I_{i(j)},\ j = 1,2,\ldots$. The processing time of instruction $I_{i(j)}$ in the cache is $\nu_{i(j)}\tau$, where $\nu_{i(j)}$ is the number of clock cycles it takes to proceed the instruction in cache in cache hit condition. The completion time of executing a cache-hit instruction is the dequeuing time from cache, that is,

$$\gamma_{i(j)} \ = \ max\{\ \alpha_{i(j)} + \nu_{j(i)}\tau,\ \delta_{i(j)-\lambda}\ \}, \tag{4.7}$$

where λ is the total number of Miss Status Holding Registers (MSHR) entries. We assume if the number of instructions in MSHR reaches λ, the whole memory system stops processing. The derivative of $\gamma_{i(j)}$ with respect to the clock frequency is given by the following equation.

$$\gamma'_{i(j)}(\tau) = \begin{cases} \alpha'_{i(j)}(\tau) + \nu_{i(j)}, & \text{if Load/Store Unit} \\ & \text{does not stop} \\ & \text{after processing} \\ & \text{instruction } I_{i(j)-1}. \\ \delta'_{i(j)-\lambda}(\tau), & \text{otherwise.} \end{cases} \tag{4.8}$$

If instruction $I_{i(j)}$ is a cache miss then it needs to access other storage devices such as DRAM. The major part of its latency can be approximated by a term denoted by $MEM_{i(j)}$, which typically is hundreds of clock cycles and hence one-to-two orders of magnitude longer than compute instructions. Note that $MEM_{i(j)}$ is independent of τ since the clock of such memory systems is different from the clock of cores and caches. The completion time of a cache-miss instruction $I_{i(j)}$ in Memory stage is its departure time from the MSHR

back to execution, and denoting it by $\delta_{i(j)}$, it can be seen from the above discussion that

$$\delta_{i(j)} \;=\; max\{\; \gamma_{i(j)} + M_{i(j)}\tau + MEM_{i(j)} \;,\; \delta_{i(j)-1}\}, \tag{4.9}$$

where $M_{i(j)}\tau$ is the proceeding time in MSHR. We derive $\delta'_{i(j)}(\tau)$ as

$$\delta'_{i(j)}(\tau) = \begin{cases} \gamma'_{i(j)}(\tau) + M_{i(j)}, & \text{if instruction } I_{i(j)-1} \\ & \text{leaves MSHR} \\ & \text{before instruction} \\ & I_{i(j)} \text{ is completed.} \\ \delta'_{i(j)-1}(\tau), & \text{if instruction } I_{i(j)-1} \\ & \text{stays in MSHR} \\ & \text{when instruction} \\ & I_{i(j)} \text{ is completed.} \end{cases} \tag{4.10}$$

Thus, the completion time of executing instruction I_i is computed as follows:

$$\beta_i = \begin{cases} \alpha_i + \mu_{(i)}\tau, & \text{if instruction } I_i \text{ is a} \\ & \text{non-memory instruction} \\ \gamma_{i(j)}, & \text{if instruction } I_i \text{ is a cache} \\ & \text{hit memory instruction} \\ \delta_{i(j)}, & \text{if instruction } I_i \text{ is a cache} \\ & \text{miss memory instruction.} \end{cases} \tag{4.11}$$

Combine equation (4.8) and equation (4.10), we have

$$\beta_i'(\tau) = \begin{cases} \alpha_i'(\tau) + \mu_{(i)}, & \text{if instruction } I_i \\ & \text{is not a} \\ & \text{memory instruction.} \\ \gamma_{i(j)}'(\tau), & \text{if instruction } I_i \\ & \text{is a cache hit} \\ & \text{memory instruction.} \\ \delta_{i(j)}'(\tau), & \text{if instruction } I_i \\ & \text{is a cache miss} \\ & \text{memory instruction.} \end{cases} \tag{4.12}$$

Finally, let us consider the final stage, Commit. The order of departure of instructions should be the same as their arrival order. Let d_i denote the time that instruction I_i in the ROB is committed (dequeuing time), then we have

$$d_i \;=\; max\{\, \beta_i + \tau \;,\; d_{i-1} + \tau \,\}. \tag{4.13}$$

Therefore, the derivative of dequeuing time for instruction i is

$$d_i'(\tau) = \begin{cases} \beta_i'(\tau) + 1, & \text{if the entry of instruction} \\ & I_i \text{ is head of the ROB} \\ d_{i-1}' + 1, & \text{if the entry of instruction} \\ & I_{i-1} \text{ still remains} \\ & \text{in the ROB} \end{cases} \tag{4.14}$$

This is a recursive equation which gives out $d_i'(\tau)$ for all $i = 1, \ldots, M$, and in particular,

we can obtain $d'_M(\tau)$. Considering all of this during a control cycle C_n comprised of M instructions, the throughput y_n is given by

$$y_n = \frac{M}{d_M}. \tag{4.15}$$

and hence,

$$y'(\tau) = -M\frac{d'_M(\tau)}{d_M(\tau)^2}. \tag{4.16}$$

This, in conjunction with equation (4.15), gives

$$y'(\tau) \;=\; \frac{1}{M}\left(\frac{y}{u}\right)^2 d'_M(\tau). \tag{4.17}$$

We use the above equations to compute the sample derivatives, $\frac{dy_n}{du_n}$ for the control parameters, the core's clock frequency.

4.3 Experiments in A Full System Cycle Level Simulator

We simulated two SPLASH-2 benchmarks, Cholesky and Ocean-nc [75] in Manifold. Cholesky is a computation intensive application while Ocean-nc is a memory intensive application. Eight cores execute the Cholesky benchmark, and eight cores execute the Ocean-nc benchmark. Each control cycle consists of 50,000 instructions, chosen to balance the settling (convergence) time with local high-frequency oscillations of the actual throughput. The frequency-range of the cores is 0.5 GHz to 5 GHz. These simulations assume that a continuous range of frequencies are feasible. We set the target throughput of each core at 4000 MIPS (Million Instruction Per Second) for Cholesky, and 1000 MIPS for Ocean-nc.

A typical simulation run for the Cholesky benchmark (chosen at random from the eight cores executing this benchmark) is shown in Figure 4.3, where the horizontal axis indicates time in ms and the vertical axis indicates instruction throughput for a single core. The

value of ξ in Equation 4.3 is $\xi = 1$. The total run time of 333 ms is the duration of the Cholesky program. We can see from the graph a fast rise in throughput from an initial value of 400*MIPS* to about 4300*MIPS* in 0.8 ms. Thereafter the throughput stabilizes at about the target value of 4,000 MIPS except for sporadic variations which are due to variable program workload and other random aspects of the system. However, the controller seems to compensate for them in short time-frames. Furthermore, the average throughput computed over the time interval [0.8*ms*,333*ms*] (soon after the throughput has reached the target value) is 3964.4 MIPS, which is quite close to the target throughput of 4,000 MIPS.

Similar results for the Ocean-nc benchmark are shown in Figure 4.4 and Figure 4.5. Figure 4.4 depicts the graph of the instruction throughput for the first 35 ms of the program, while Figure 4.5 shows the throughput for the entire run of 333 ms. The reason for restricting the results to a subset of the program's duration is that the graph shows the rapid convergence of the throughput from its initial value of 600*MIPS* to about the target value at time 0.5 ms, which is not visible in Figure 4.5. In both parts of the figure we discern fluctuations of the throughput from its target value, but the control algorithm stabilizes the throughput rapidly. These fluctuations (oscillations) are more pronounced than in the results concerning the Cholesky benchmark; the reason is that Ocean-nc is more memory intensive than Cholesky, hence it experiences wider load variations. As mentioned earlier, the parameter $\xi \in (0, 1)$ in Equation (4.3) can be used to reduce oscillations in the throughput profile. To test this point we simulated the control algorithm with $\xi = 0.2$ for both the Cholesky and Ocean-nc benchmarks. The results, shown in Figure (4.6) and Figure (4.7), respectively, exhibit fewer and smaller oscillations but larger settling times as compared to the respective results in Figure (4.3) and Figure (4.4), resp., where $\xi = 1.0$. This is not surprising in light of the fact that the controller's gain is smaller. In fact, the measured average throughput for Choleaky is 3989.8 MIPS for $\xi = 0.2$ and 3964.4 MIPS for $\xi = 1.0$, whereas for Ocean-nc, it is 1004.6 MIPS for $\xi = 0.2$ and 1008.9 MIPS for $\xi = 1.0$.

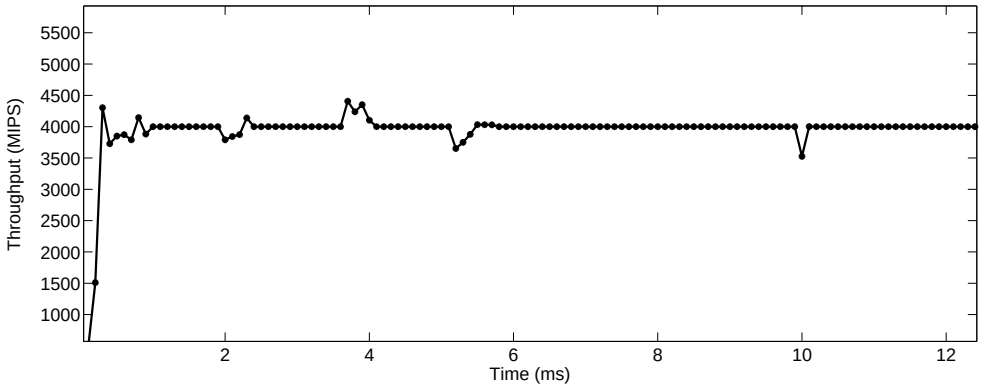

Figure 4.3: Throughput regulation: Cholesky

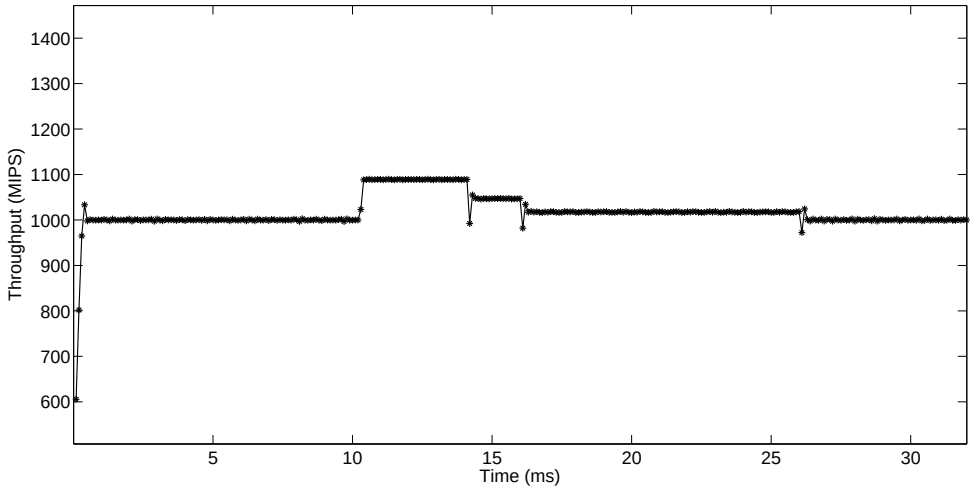

Figure 4.4: Throughput regulation: Ocean-nc (beginning part)

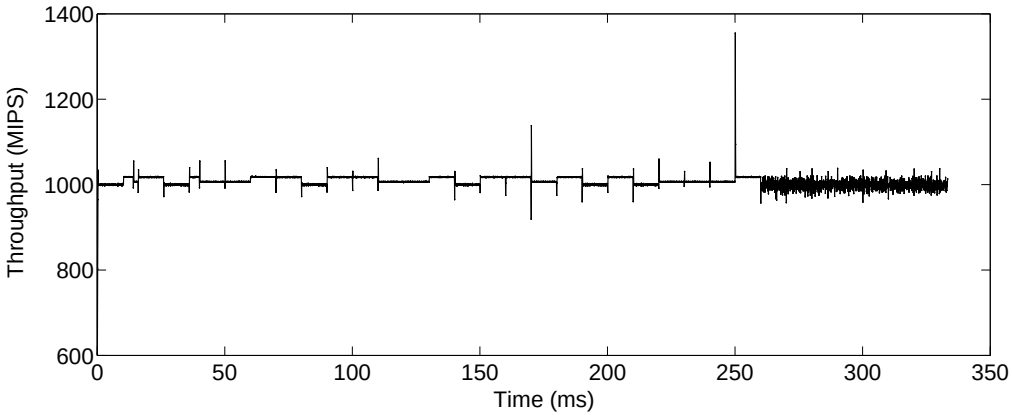

Figure 4.5: Throughput regulation: Ocean-nc (full execution)

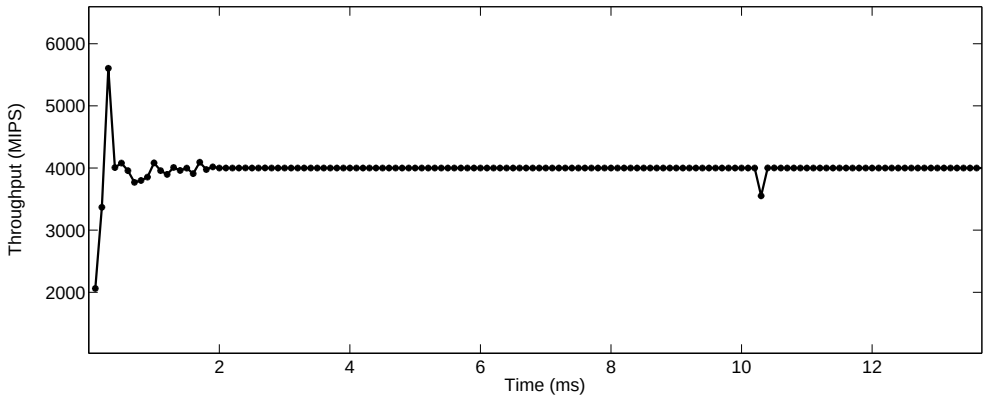

Figure 4.6: Throughput regulation (modified algorithm): Cholesky

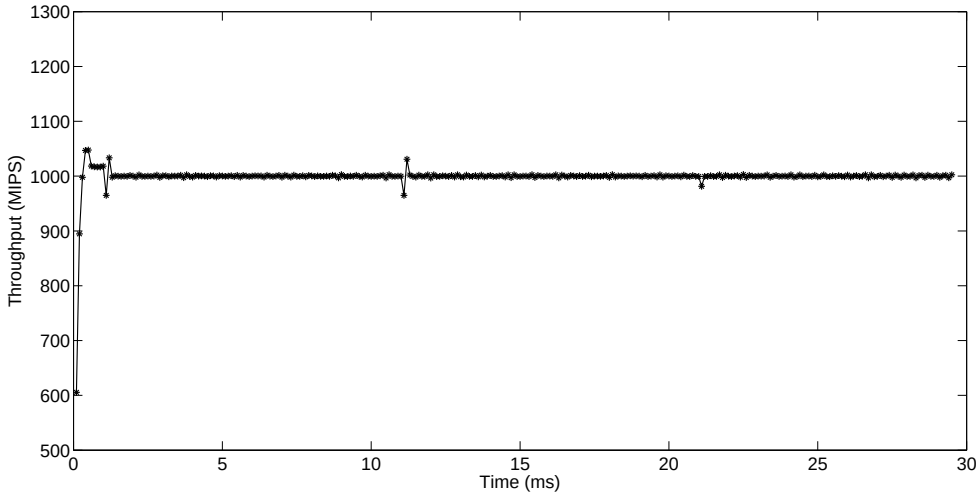

Figure 4.7: Throughput regulation (modified algorithm): Ocean-nc

4.4 Implementataion on an Intel Haswell 4-Core Processor

4.4.1 Modified Regulator

In this section we report on our design implemented in an Intel Haswell processor. We assume each core can share its execution resources with up to two threads. Therefore, all threads executing on the processor share the same frequency as the multi-core processor. The goal of our design is to achieve thread-level throughout regulation as well as processor-level throughput regulation. Since four cores are residing on one voltage island, they share the same operating frequency. Hence, for the throughput regulation on Haswell processor,

we introduce a centralized regulator that uses the feedback control described in section 4.1. The control variable in this section is the processor's operating frequency. The input for the controller is the target throughput for each thread. More specifically, we assume all threads have the same target throughput, and this target throughput is the input for the centralized regulator. The operating frequency is computed based on the average throughout of executing threads. In other words, if a thread stays idle during certain control cycles, the thread's throughput is not taken into account when calculating the average throughput for those control cycles. The sample path gradient is approximated by the value of average thread throughput over operating frequency. The flow chart is shown in Figure (4.8).

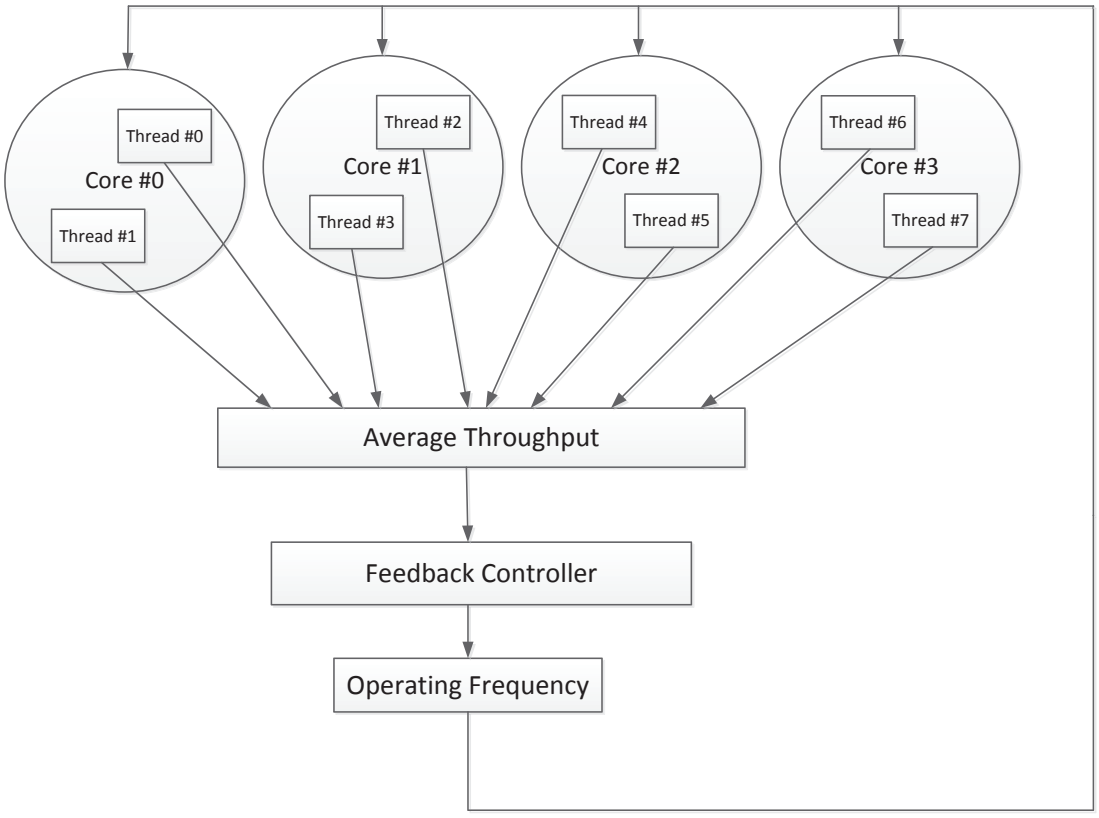

Figure 4.8: Flow Chart of the implementation in Centralized Controller

The integral gain of the regulator, A_{n+1} in equation 4.1 at control cycle C_n is approxi-

mated by

$$A_{n+1} = \frac{T_n}{u_n} \tag{4.18}$$

where T_n is the average throughput of running threads at control cycle C_n, and u_n is the operating frequency of the processor at control cycle C_n. e_n in equation 4.2 in this section is the difference between the average thread throughput and the target thread-level throughput.

Haswell processors employ 16 discrete frequency levels, ranging from $0.8GHz$ to $3.4GHz$. Therefore, we map the continuous frequencies computed to discrete frequencies of Haswell processors. The mapping table is shown in Table (4.1). In control cycle C_n, the controller completes the follow steps.

1. Collect throughput of each executing thread.
2. Compute the average throughput of all the executing threads, denoted as y_{n-1}.
3. Compute the adaptive gain A_n by the approximation $A_n = \frac{y_{n-1}}{u_{n-1}}$.
4. Apply the feedback control law and achieve the frequency u_n by equation 4.1.
5. Choose the actual operating frequency for control cycle C_n by the mapping table 4.1.

Table 4.1: Continuous-Discrete Frequency Mapping Table

Frequency Computed (GHz)	Frequency Set (GHz)	Frequency Computed (GHz)	Frequency Set (GHz)
<0.9	0.8	2.1 - 2.3	2.2
0.9 - 1.05	1.0	2.3 - 2.45	2.4
1.05 - 1.2	1.1	2.45 - 2.6	2.5
1.2 - 1.4	1.3	2.6 - 2.8	2.7
1.4 - 1.6	1.5	2.8 - 3.0	2.9
1.6 - 1.75	1.7	3.0 - 3.15	3.1
1.75 - 1.9	1.8	3.15 - 3.3	3.2
1.9 - 2.1	2.0	>3.3	3.4

4.4.2 Experimental Results

We first test Barnes benchmark from Splash-II benchmark suits. For the throughput target of 1,200 MIPS, the results are shown in Figure 4.9, where the horizontal axis indicates time in ms and the vertical axis indicates instruction throughput. The total run time is 100 ms, and it corresponds to about 1,000 control cycles. The throughput rises from an initial value of 739.2 MIPS to the target level of 1,200 MIPS in about 1.3 ms, or 13 control cycles. The average throughput computed over the time interval [13ms,100ms] (soon after the throughput has reached the target value) is 1,166.5 MIPS, which is 33.5 MIPS off the target level of 1,200 MIPS. The graph of the frequencies is shown in Figure 4.10, and it indicates no saturation throughout the program. We partly attribute the gap to the quantization error due to the rounding off of the frequencies to their nearest values in W, which is evident from Figure 4.10.

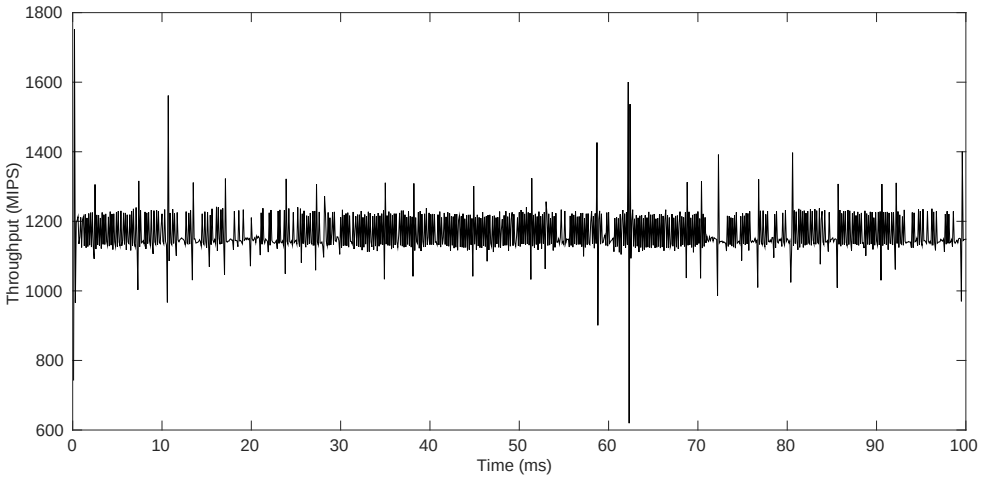

Figure 4.9: Barnes: throughput vs. time, target = 1,200 MIPS

For the target level of 1,000 MIPS, see Figure 4.11 and Figure 4.12 the throughput climbs from its initial value of 633.2 MIPS to its target level in 1.5ms, or 15 control cycles. There was no frequency saturation, and the average throughput in the [1.5ms, 330ms] interval is 990.6, which means an offset of 9.4MIPS from the target level of 1,000 MIPS.

For the target level of 800 MIPS, see Figure 4.13 and Figure 4.14 the throughput climbs

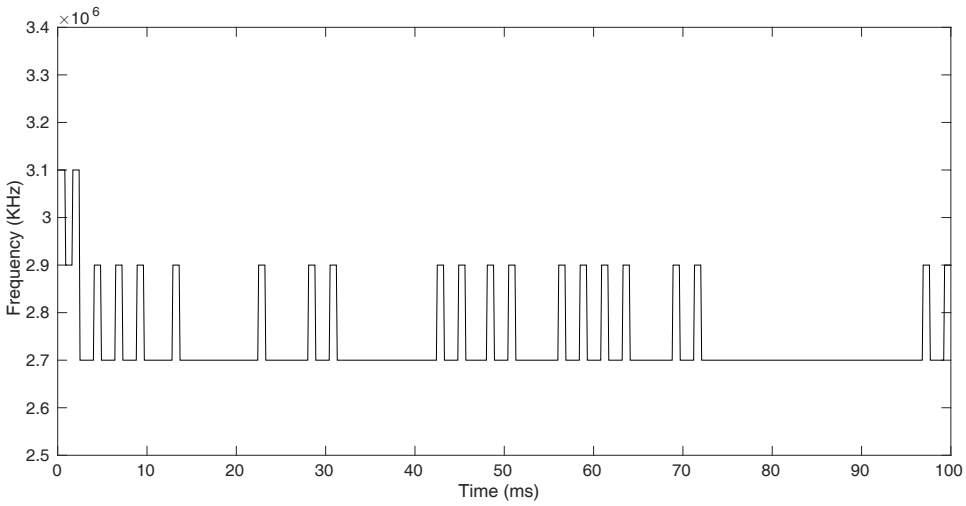

Figure 4.10: Barnes: frequency vs. time, target = 1,200 MIPS

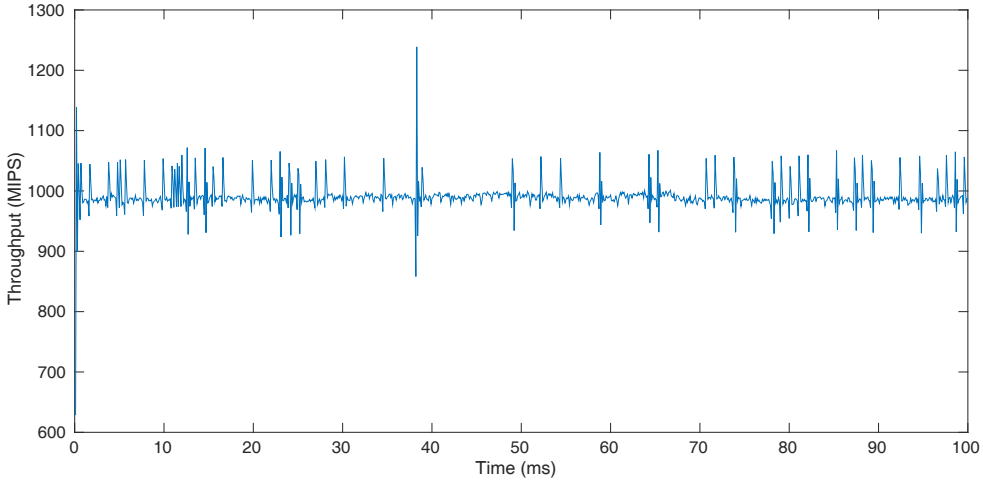

Figure 4.11: Barnes: throughput vs. time, target = 1,000 MIPS

from its initial value of 763.1 to the target level in 1.0 ms, or 10 control cycles. There was no frequency saturation, and the average throughput is 829.7 MIPS, which is 29.7 MIPS off the target level of 800 MIPS.

Recall that we proposed a way to reduce the throughput oscillations and frequency saturation by modifying the control algorithm, by replacing ξ in Equation 4.3 with 0.2. Although this worked well for the Manifold simulation with the throughput target of 1,200 MIPS, it yielded poor results for the Haswell implementation. After a few iterations the

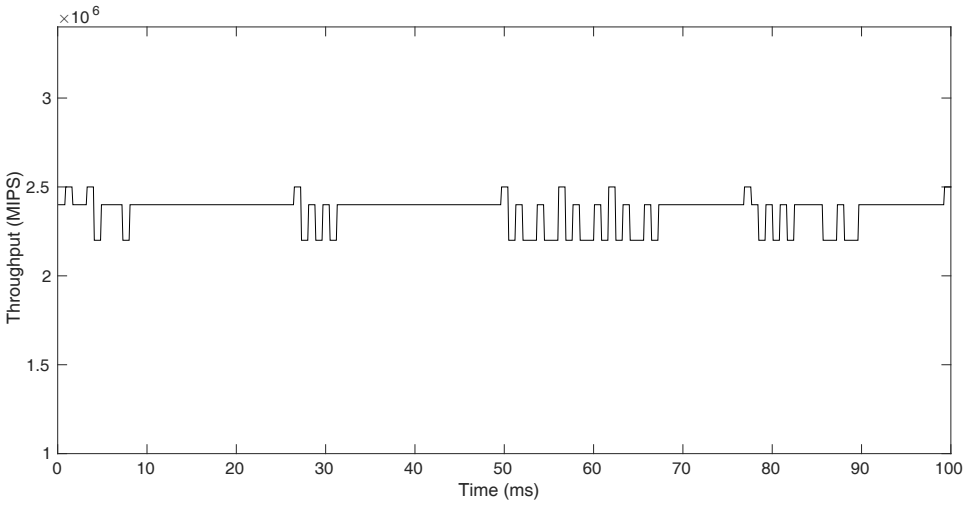

Figure 4.12: Barnes: frequency vs. time, target = 1,000 MIPS

processor frequency was trapped at a value. The reason is in the quantization error inherent in the algorithm, which is due to the rounding off of the computed control variable. The step size for modifying the control variable is insufficient to take that variable out of its current value.

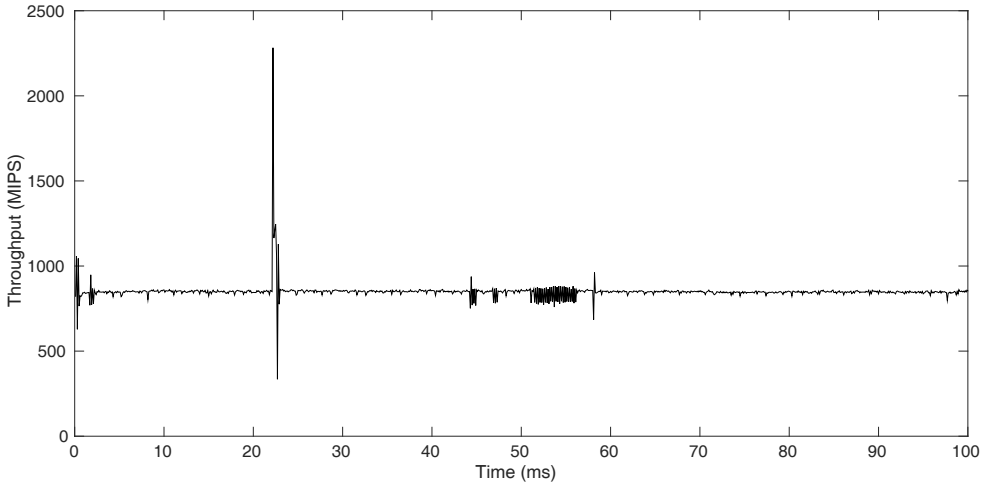

Figure 4.13: Barnes: throughput vs. time, target = 800 MIPS

We also chose two applications: Depth-First Search (DFS) and Connected Component from the GraphBig suite of benchmarks [77]. DFS is a fundamental graph analysis primitive in many graph analytics applications, containing tree searching and graph data-

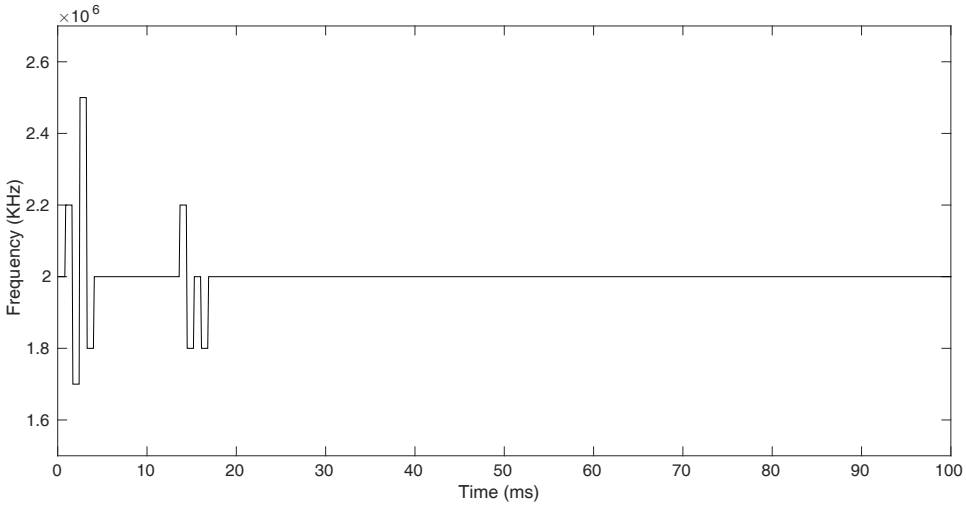

Figure 4.14: Barnes: frequency vs. time, target = 800 MIPS

structure search algorithms. Connected Component is a graph topology analysis tool also found in many high level graph analytics applications. A complete implementation of these programs typically has two major elements: data generation associated with an application, and the computations on those data. In a benchmark evaluation, the element that generates the data may itself require considerable computing times impacting the overall performance. We present the results for first 600ms. Each control cycle of the regulation algorithm lasts 0.1 ms, hence the various graphs depict the systems response for 6,000 cycles. Recall that the Haswell machine by which we execute the regulation technique has a discrete set of 16 frequencies in the range 0.8 GHz to 3.4 GHz. Whenever the controller attempts to assign a frequency outside that range when saturation occurs. As the fact that we consider cloud computing applications, the GraphBig programs are more memory-intensive than the Splash 2 programs, and they also make memory calls throughout their duration and not only at their initial stages. For these reasons we expect longer throughput rise times and larger oscillations. However, we do not attempt to test the effects of saturation. Rather, we test the control technique at a range of throughput setpoints where no saturation occurs, and as a result, obtain better average tracking than those presented for the Splash-2 programs.

Target value of 1200 MIPS The results for throughput regulation of the DFS benchmark are shown in Figure 4.15. The throughput rises from an initial value of 794.51 MIPS to the target level of 1,200 MIPS in about 19 ms. The average throughput computed over the time interval [19ms, 600ms] is 1,219 MIPS, which is 19 MIPS more than the target level of 1,200 MIPS. The graph of the frequencies are shown in Fig 4.16, and they indicate considerable saturation at the lower frequency of 0.8 Ghz. In spite of that, the average-throughput offset from its target level is quite small, and symptomatic of experiments without frequency saturation. For the Connected Component benchmark, the graph of the throughput is shown in Figure 4.17. The throughput rises from an initial value of 876.39 MIPS to its target value of 1,200 MIPS in about 19 ms. The frequency-graph, shown in Figure 4.18, indicates some saturation at the lower boundary 0.8 GHz. The average throughput in the interval [19ms, 600ms] is 1211.4 MIPS, which is 11.4 MIPS over the target level of 1,200 MIPS. Frequency saturation has provided the main argument for explaining large tracking errors, and hence we presented the frequency graphs obtained from the experiments described thus far. The forthcoming experiments incurred no frequency saturation and therefore, and in order to limit the length of the paper, we omit the frequency graphs.

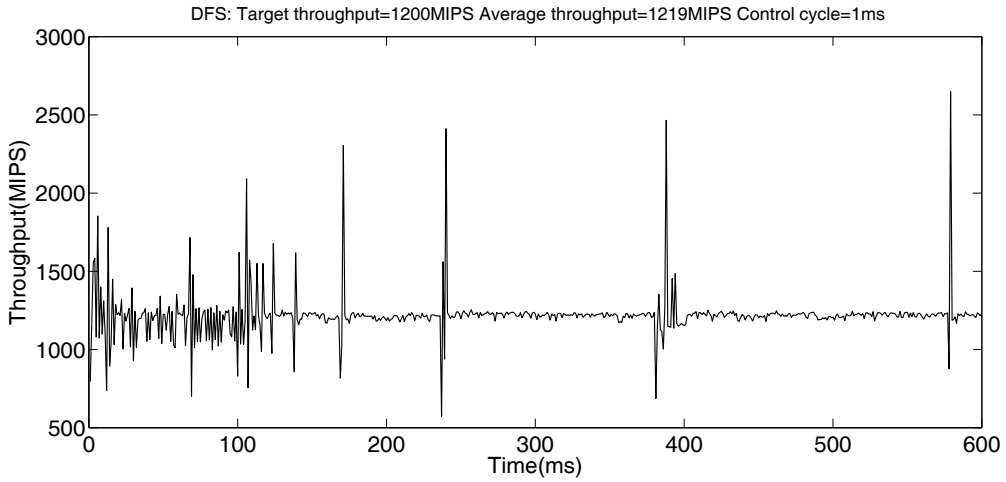

Figure 4.15: DFS: throughput vs. time, target = 1,200 MIPS

Target value of 1500 MIPS For the DFS benchmark, the graph of the throughput is depicted in Figure 4.19. The throughput rises from its initial value of 808.493 MIPS to the

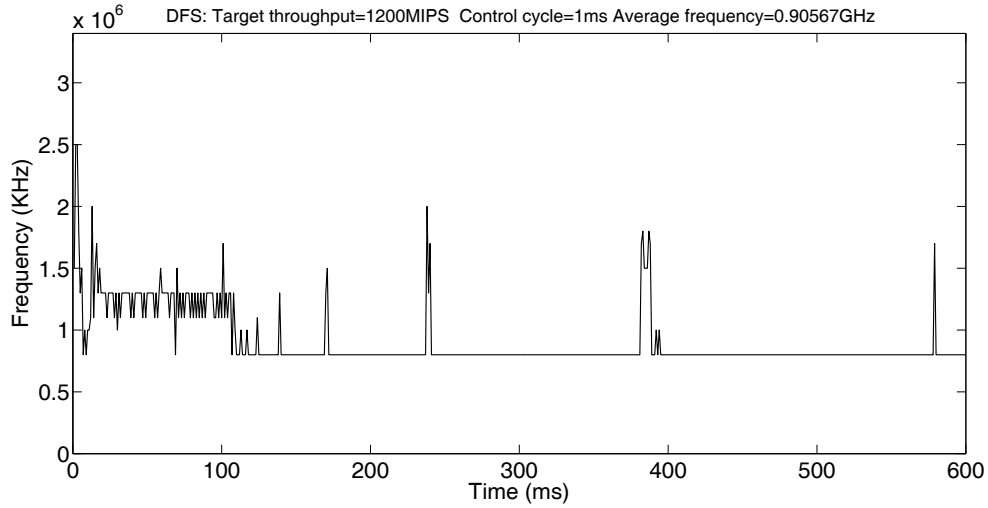

Figure 4.16: DFS: frequency vs. time, target = 1,200 MIPS

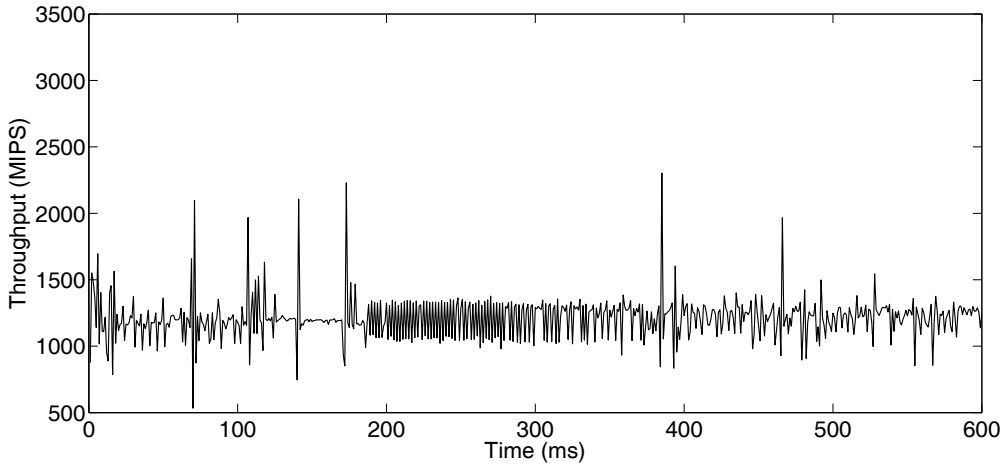

Figure 4.17: Connected Component: throughput vs. time, target = 1,200 MIPS

target level of 1,500 MIPS in 17 ms, or 170 control cycles. The frequency-graph, shown in Figure 4.20, indicates saturation only at three points, which is negligible. The average throughput is 1545.5 MIPS, which is 45.5 MIPS more than the target level of 1,500 MIPS.

For Connected Component, the throughput graph is depicted in Figure 4.21, while the graph of frequency shown in Figure 4.22, indicates no saturation. The throughput rises from an initial value of 1073.18 MIPS to its target level in 16 ms, or 160 control cycles. The average throughput is 1486.2 MIPS, which is 13.8 MIPS below its target level.

Target value of 1900 MIPS For the DFS benchmark, the graph of the throughput is

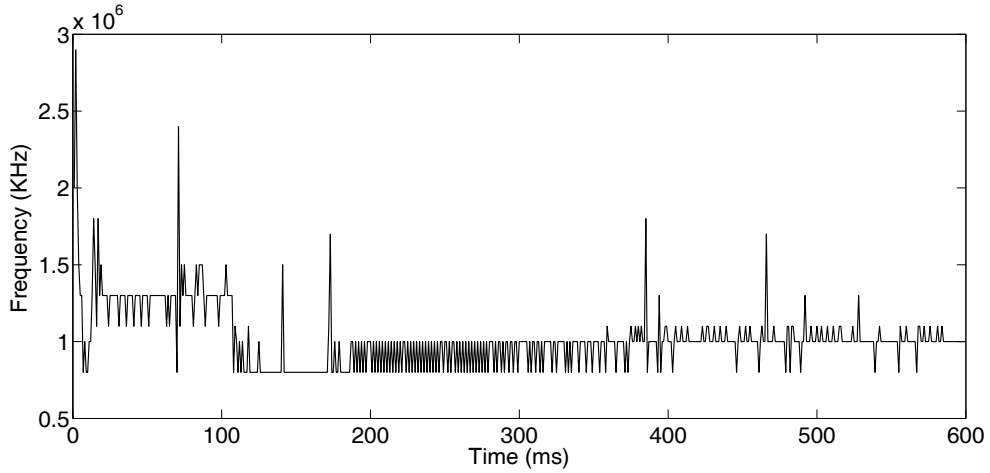

Figure 4.18: Connected Component: frequency vs. time, target = 1,200 MIPS

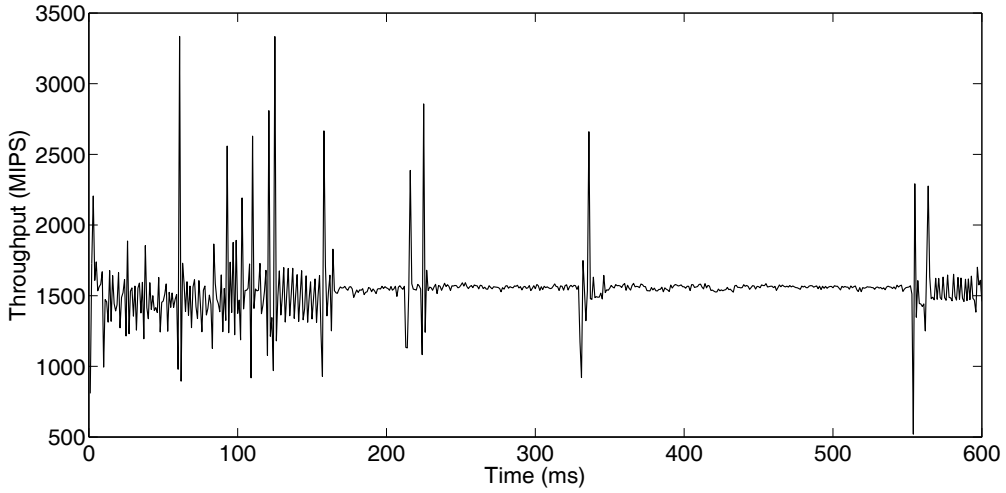

Figure 4.19: DFS: throughput vs. time, target = 1,500 MIPS

shown in Figure 4.23. The throughput rises from its initial value of 1286.04 MIPS to the target level of 1,900MIPS in 11 ms, or 110 control cycles. There is no frequency saturation, and the average throughput is 1918.6 MIPS, which is 18.6 MIPS over than the target level of 1,900 MIPS.

For the Connected Component benchmark, the graph of the throughput is shown in Figure 4.24. The throughput rises from an initial value of 901.288 MIPS to its target value of 1,900 MIPS in about 18 ms, or 180 control cycles. The is no saturation except at a single point. The average throughput in the interval [18ms,600ms] is 1892.0 MIPS, which is 8.0

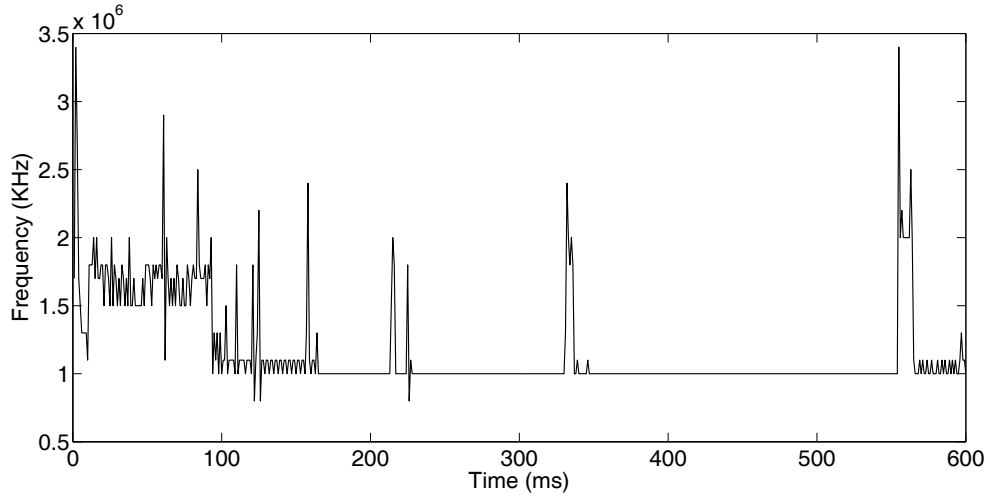

Figure 4.20: DFS: frequency vs. time, target = 1,500 MIPS

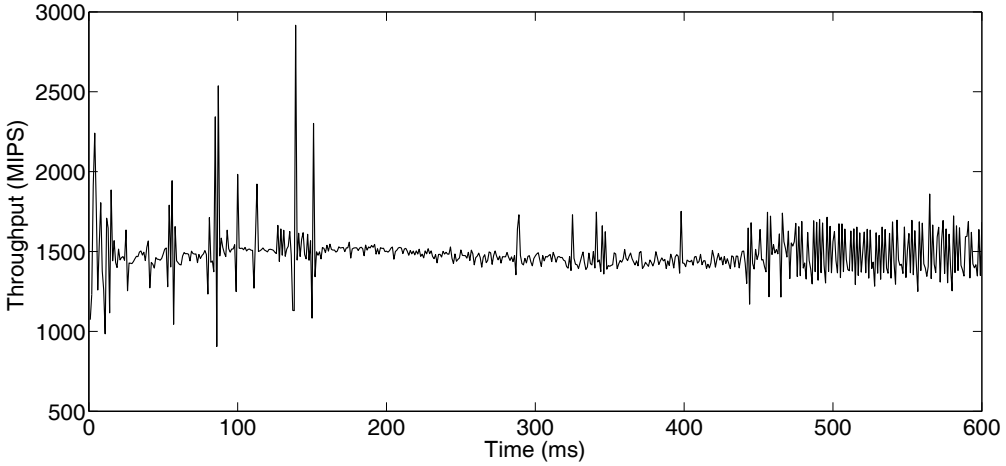

Figure 4.21: Connected Component: throughput vs. time, target = 1,500 MIPS

MIPS off the target level of 1,900 MIPS.

In addition to the reasons mentioned above, there are several other factors that may induces tracking errors. On observation that the big sparks in throughput appears periodically implies that system calls from the operating system interrupts the tracking. We can also observe that the fluctuation in the throughput of GraphBig benchmarks are bigger in the beginning part of execution compared to the rest of the execution. A complete implementation of GraphBig programs typically has two major phases: data generation associated with an application, and the computations on those data. During the first phase,

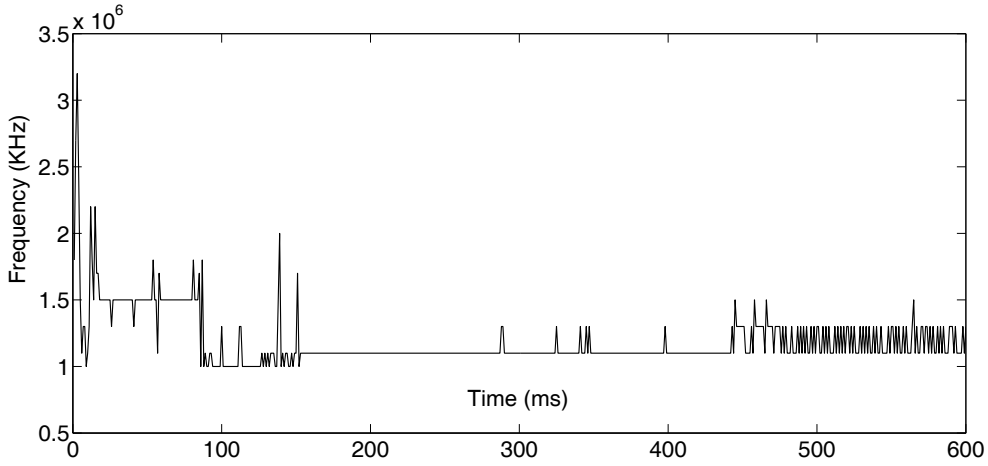

Figure 4.22: Connected Component: frequency vs. time, target = 1,500 MIPS

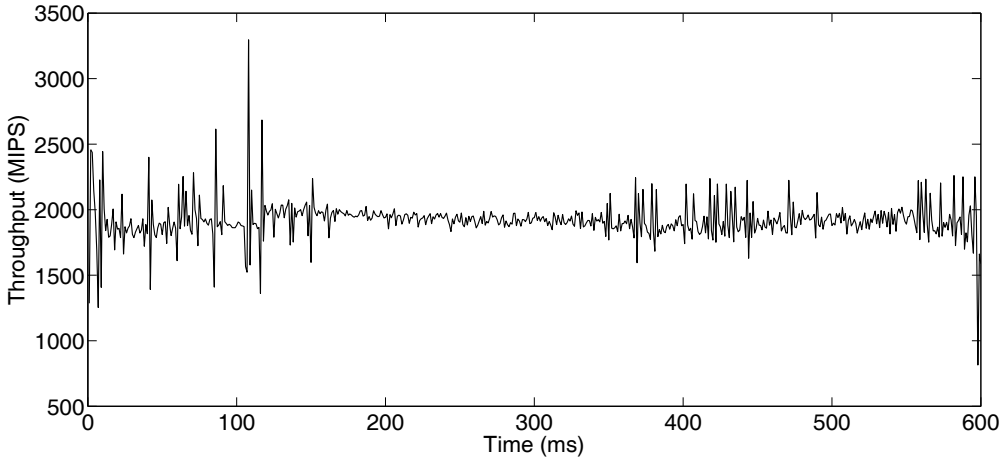

Figure 4.23: DFS: throughput vs. time, target = 1,900 MIPS

there are more memory participation compared to the second phases, which explains bigger fluctuation in the beginning of the program execution.

4.5 Concluding Remarks

This section describes the design and test of a technique for regulating instruction throughput in computer processors by adjusting the clock frequencies at the cores as well as the processor. The tests, applied to several industry-benchmark programs, were performed in a full system cycle level simulator as well as an Intel 4-core Haswell processor. The regulator

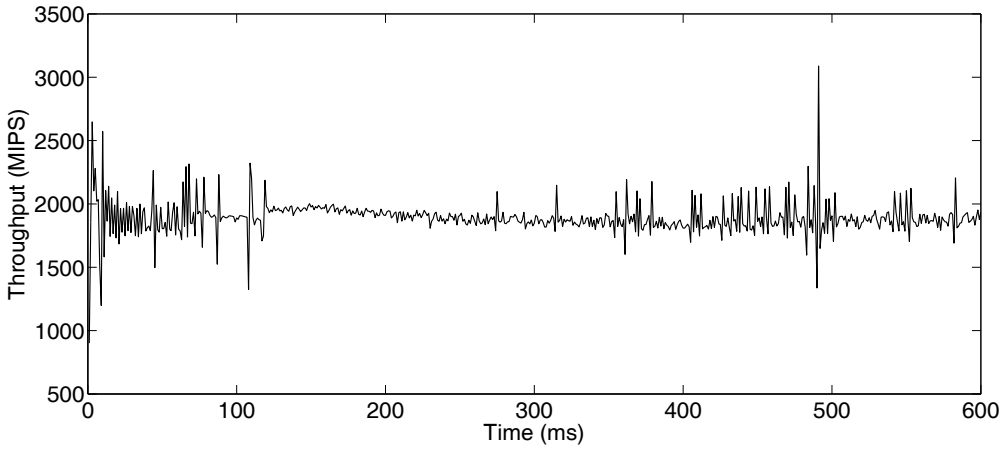

Figure 4.24: Connected Component: throughput vs. time, target = 1,500 MIPS

is based on a standalone integrator with a variable gain, adjusted in real time by estimating the IPA derivative of the plant function. The tracking convergence of the closed loop system is robust with respect to computational errors, which suggests that the precision of computations in the loop may be reduced in order to speed up their rates. The regulation technique performs well in various industry benchmarks, especially in cloud computing applications which include significant memory instructions and therefore are generally hard to control.

CHAPTER 5

POWER REGULATION FOR MULTICORE PROCESSORS

This chapter addresses the problem of power regulation where the power consumption of a multi-core processor is maintained at a set target by varying processor frequencies. In multi-core processors, the relationships between workloads, power dissipation, resulting thermal fields, and their interactions with the leakage current are complex. For example, application workloads exhibit time-varying computation and memory access behaviors resulting in spatially and temporally varying power dissipation and non-uniform thermal fields. The cross-chip variations in temperature coupling with circuit leakage and delay increases full-chip leakage power. As a result, the peak throughput is decreased and chip reliability is degraded. Thus, effective control of power dissipation is critical to the reliable and high performance operation of multi-core processors.

A general technique for controlling power and temperature is based on setting the appropriate power state of voltage islands, which is Dynamic Voltage Frequency Scaling (DVFS). The development of effective controls based on DVFS faces several challenges. First, the relationship between the clock frequency and core power is complicated by other factors such as the coupling between temperature and leakage power. Second, application workloads have time varying compute and memory system behaviors requiring a robust and adaptive control strategy to manage power dissipation. Third, distinct cores in a voltage island execute distinct instruction streams with distinct behaviors but may share a common clock frequency. For example, the Intel Haswell processor tested in this work has four cores sharing a single voltage island and executing eight threads at the same voltage and frequency [71]. This chapter presents a feedback power regulator using an on-line model for estimating power-frequency relationship by a least-square system identification method.

The main contributions in this chapter are:

- A novel adaptive gain feedback regulator for power regulation in multi-core processors by DVFS with accuracy and stability. The regulator does not depend on the application and can dynamically adjust to various workloads.

- A power model based on system identification methods for estimating frequency and power relationship.

- An implementation and testing of the design on an Intel Haswell 4-core processor with industry standard benchmarks as well as cloud computing benchmarks.

5.1 A Power Model and a Power Regulator

In this section, we adapt the feedback control law presented in Chapter 4 to regulate the power of the multicore processors. The A_{n+1} in Equation 4.1 becomes the gradient of power with respective to processor frequency in this Chapter. In this section, we develop a dynamic power model using system identification techniques to obtain the value of A_{n+1}.

Consider a processor driven by a supply voltage V and operating at a frequency u, the total power P_{total} dissipated is composed of static power P_{static} and dynamic power $P_{dynamic}$ as shown in equation 5.1

$$P_{total} = P_{dynamic} + p_{static} \tag{5.1}$$

Dynamic power depends on supply voltage and clock frequency of the processor, while static power depends on supply voltage, temperature, and manufacturing technology parameters, see [82]. The dynamic power has the following form

$$P_{dynamic} = \alpha * C * V^2 * u \tag{5.2}$$

where α is a time-varying workload parameter representing the switching activity of the transistor gates, and C is the switching capacitance of the processors. The static power

depends on the supply voltage and temperature, on the other hand the temperature depends on the total power [82]. For the experimented processor, the Intel Haswell processor, the supply voltage V cannot be measured directly. Furthermore, we can only measure total power consumption, not static power and dynamic power separately. In other words, we can measure the operating frequency and total power of the processor but not the supply vo ltage. Therefore, we developed a power model using system identification approaches to describe the relationship between power and frequency without knowing the supply voltage and manufacturing factors. In our case, the power consumption of the controlled processor varies during the program run depending on the program workload. Therefore, dynamically adjusting the model according to the runtime information is essential in order to keep online tracking of the target power. On the other hand, rapid response of the system is required for real time tracking, which means the complexity of the model should be designed to guarantee limited computation time. Otherwise the delay caused by computing the model will affect tracking accuracy. Therefore, we apply the system identification approach to dynamically model the relationship between power and frequency of the controlled processor.

We estimate the power and frequency relationship as a third-order polynomial. A similar third-order polynomial model has been explored by researchers to predict processor power [83]. Next, we use system identification techniques to demonstrate the model online.

The clock frequency of the processor in control cycle c_n, $n = 1, 2, \ldots$, is denoted as u_n, and the power consumption of the processor is denoted as p_n. Hence, the relationship between power and frequency can be modeled as the polynomial shown in equation 5.3 [3].

$$p_n = a_n \cdot u_n^3 + b_n \cdot u^2 + c_n \cdot u + d_n \tag{5.3}$$

where a_n, b_n, c_n, d_n are parameters that is determined by runtime workload, temperature, leakage current and manufacturing factors.

Let's define

$$x_i = \begin{bmatrix} (u_i)^3 & (u_i)^2 & u_i & 1 \end{bmatrix}$$

$$w_n = \begin{bmatrix} a_n & b_n & c_n & d_n \end{bmatrix}$$

Hence,

$$p_n = w_n \cdot x_n^T \tag{5.4}$$

Applying least square method, the W matrix is updated in each control cycle by the following equation:

$$W = (X^T X)^{-1} \cdot X^T \cdot P \tag{5.5}$$

However, the complexity to solve equation 5.5 is $O(n^3)$. The long time it takes for the processor to complete computation will cause delay in real time tracking system. To reduce the complexity, we use recursive least square method with statistical learning models, where the data samples are assumed to be independent and identically distributed random variables. Since we assume noise is iid (independently identically distributed), with zero mean, Gaussion distribution, the covariance matrix is identity. Hence, we have:

$$z_n = x_n \cdot x_n^T \tag{5.6}$$

$$z_0 = I^{4*4} \tag{5.7}$$

$$z_n = z_{n-1} - \frac{z_{n-1} \cdot x_n \cdot {x_n}^T \cdot z_{n-1}}{1 + {x_n}^T \cdot z_{n-1} \cdot x_n} \tag{5.8}$$

Remember X is $n * 4$, and W is $4 * 1$. The parameters are computed by the following

recursive equation,

$$w_n = w_{n-1} - z_n \cdot x_n \cdot (x_n^T \cdot w_{n-1} - p_n) \tag{5.9}$$

The complexity is now reduced to $O(n^2)$. For implementation on Intel Haswell machine, we only need to store the value of z_n and update it at each control cycle.

According to the power-frequency relationship, we can compute the derivative of power with respect to frequency as follows,

$$\frac{dp_n}{du} = 3 \cdot a_{n-1} \cdot u_{n-1}^2 + 2 \cdot b_{n-1} \cdot u_{n-1} + c_{n-1} \tag{5.10}$$

5.2 Implementation in an Intel Haswell 4-core Processor

We implement our design in the Intel Haswell processor introduced in Chapter 3. We test the proposed power regulator with Splash II [75] benchmarks and GraphBig benchmarks [77]. In this section, we present online power tracking results for Barnes and Triangle Count. The energy consumed during control cycle is measured by from RAPL "PP0_Energy:PACKAGE0" [72]. The unit for energy is J. We obtain the power by dividing total energy consumed by the whole voltage island during the control cycle by the control cycle duration time. The per-core power is obtained by dividing the power by the number of cores.

Barnes experiments The power-target value for Barnes is $10W$. For the control cycle of 10 ms the results are shown in Figure 5.1. The horizontal axis indicates time in ms and the vertical axis indicates power. The total run time is around 10000 ms, corresponding to about 1000 control cycles. The power rises from an initial value of 1.48634 W, and following a period of transient behavior lasting $780ms$, taking 78 control cycles, it settles into an oscillatory behavior about the target value of 10 W. The average power computed over the interval $[780ms$ to $10,000ms]$ (after the power has settled for the first time) is

9.986 W, which is 0.014 W below the target level of 10 W. The graph of the frequencies are shown in Figure 5.3. Throughout most of the first half of the run we notice frequency oscillations between $1.7KHz$, $1.8KHz$, and $2.0KHz$, while throughout most of the second half, the oscillations are between $1.7Khz$ and $1.8Khz$. These oscillations are due in part to the fact that the frequency-set is finite, they induce (in part) the power oscillations shown in Figure 5.1. The larger oscillations in the first half likely are due to the fact that early in the program there are more memory instructions than in the second half, where most instructions are computational. Memory instructions can take one-to-two orders more time than computational instructions, and hence having more memory instructions is associated with greater variability in in the program workload and hence in larger changes in both the control variable (frequency) and the controlled variable (power). To show the initial transient in greater detail, we depict the graph of power vs. time only for the first 1,000ms in Figure 5.2.

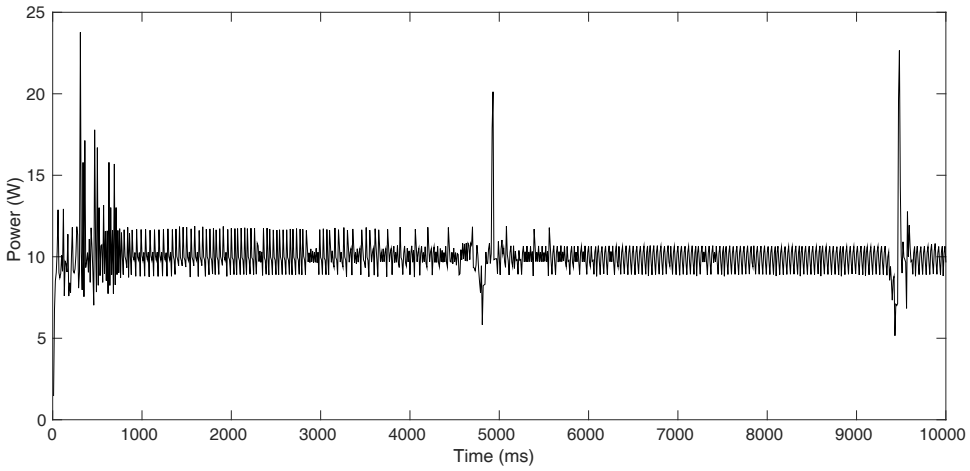

Figure 5.1: Barnes: power vs. time, target = 10 W, control cycle = 10 ms

For the control cycle of $20ms$, the graphs of power vs. time is depicted in in Figure 5.4, while the frequency graph is similar to that shown in Figure 3 for the $10ms$-cycle, hence not shown. The power rises from an initial value of 6.5511 W towards its its target value of 10 W, about which it settles in an oscillatory behavior after 860 ms, or 43 control cycles.

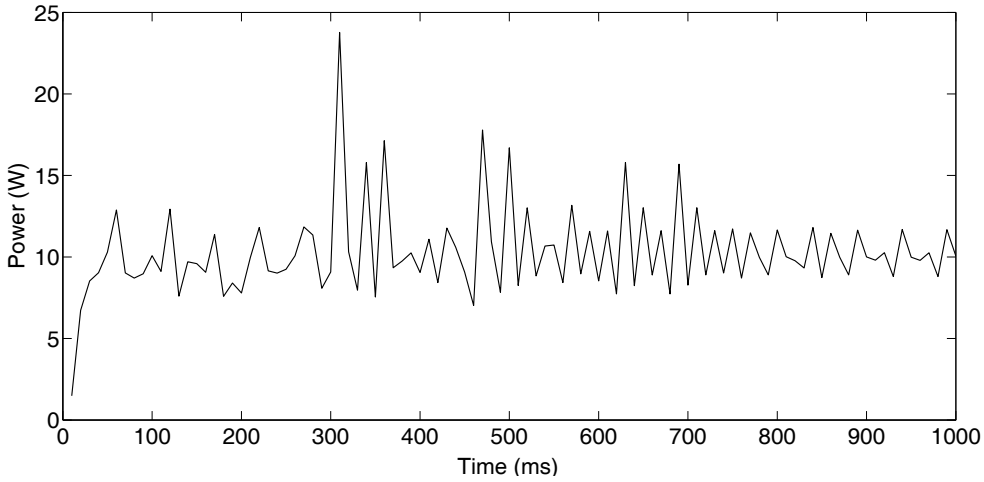

Figure 5.2: Barnes: power vs. time, target = 10 W, control cycle = 10 ms, first 1000 ms

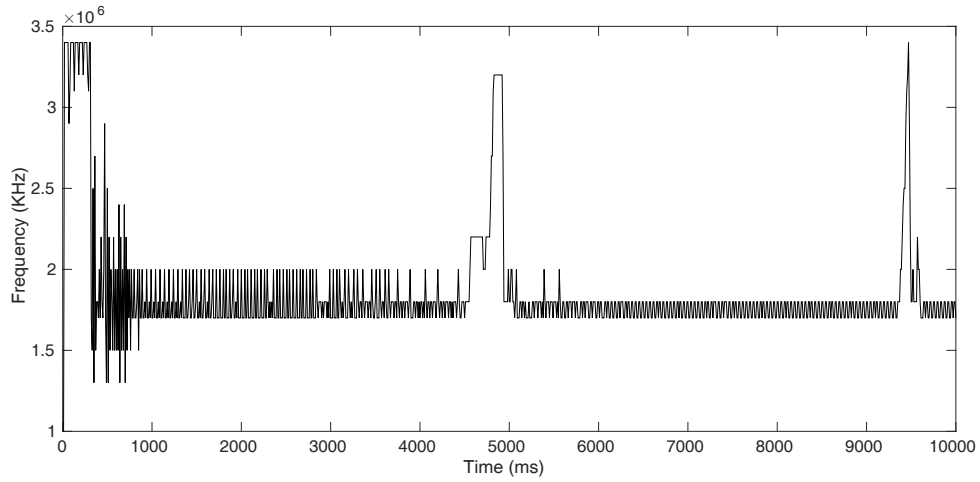

Figure 5.3: Barnes: clock frequency vs. time, target = 10 W, control cycle = 10 ms

The average power in the interval [860ms to 10,000ms] is 10.0715 W, which is 0.0715 W over the target target level of 10 W. The average frequency is 1.8291 GHz.

For the control cycle of 30*ms*, the graph of power vs. time is depicted in in Figure 5.5. The power rises from an initial value of 2.62598 W, and after 930ms (or 21 control cycles) it settles around the target value of 10 W. Its average in the interval [930,10,000] ms is 10.1821 W, which is 0.1821 W over the target level of 10 W. The average frequency is 1.8595 GHz.

In addition to the factors mentioned above, runtime errors are introduces by other re-

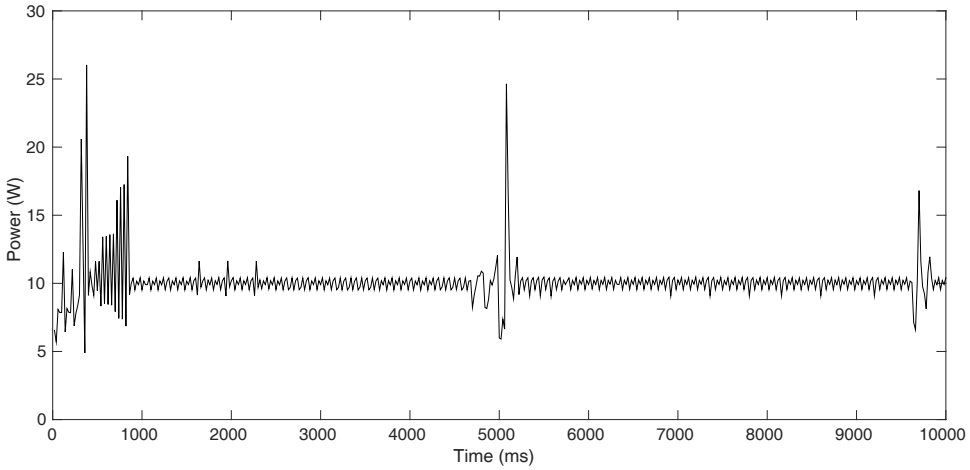

Figure 5.4: Barnes: power vs. time, target = 10 W, control cycle = 20 ms

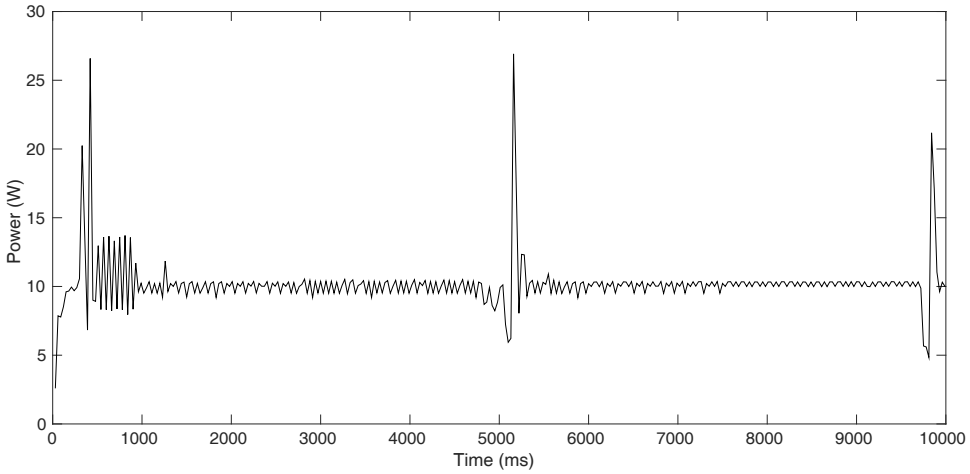

Figure 5.5: Barnes: power vs. time, target = 10 W, control cycle = 30 ms

sources during the power regulation process. We observe that in the power figures, there are big spark (sudden power consumption jump) appear periodically throughout the program run. One possible cause of that is the system calls from operating system. The operating system interrupts CPU periodically. Those system calls introduces errors in the power regulation. Furthermore, the power consumption of hardware execution units is unpredictable. Even the same kind of instructions may consume different power in the same compute units. Those various power changes introduce uncertainties in the power regulation.

Table 5.1 summarizes the time it takes the power to settle about its target value for

the first time, as well as the absolute value of the error between the average power and the btarget value, for the three control cycles of 10ms, 20ms, and 30ms. It is evident that smaller control cycles yield better results, but the differences between the results for 10ms and 30ms are minor. In all cases the regulation algorithm provides tracking in short times.

Table 5.1: Barnes: average power at different control cycles

Control Cycle (ms)	10	20	30
Error (W)	0.014	0.0715	0.1821
Settling Time (ms)	780	860	930

Triangle Count Experiments The target-power level for Triangle Count is 5 W. Note that it is lower than the target for Barnes, and the reason is that Triangle Count has considerably more memory access than Barnes, which tend to be low-frequency, low-power operations.

For a 10*ms* control cycle, the results are shown in Figure 5.6 and 5.7. The power vs time graph is depicted in Figure 5.6. The power starts at the initial value of 6.5131 W, and following an initial transients lasting 80 ms (or 8 control cycles) it settles about the target value of 5 W. The average power in the interval [80,3,500] ms is 4.9947 W, which is 0.0053 MIPS less than the target level of 5 W. The frequency graph is depicted in Figure 5.7.

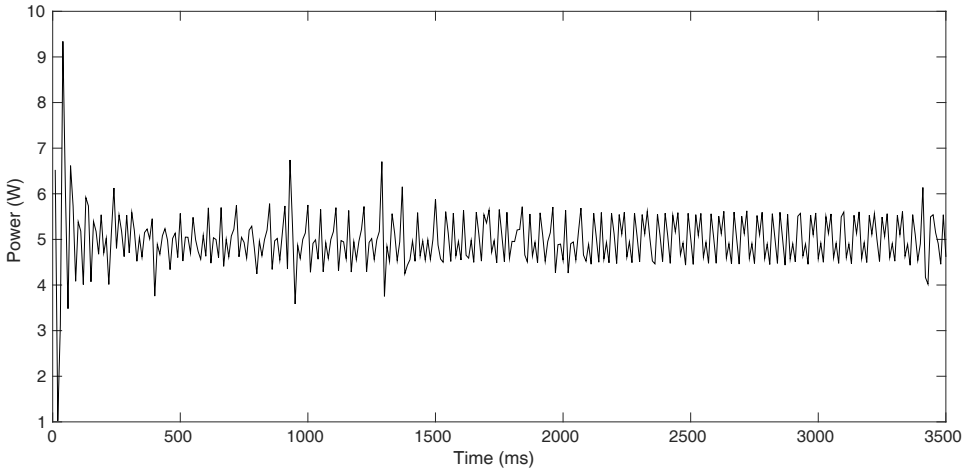

Figure 5.6: Triangle Count: power vs. time, target = 5 W, control cycle = 10 ms

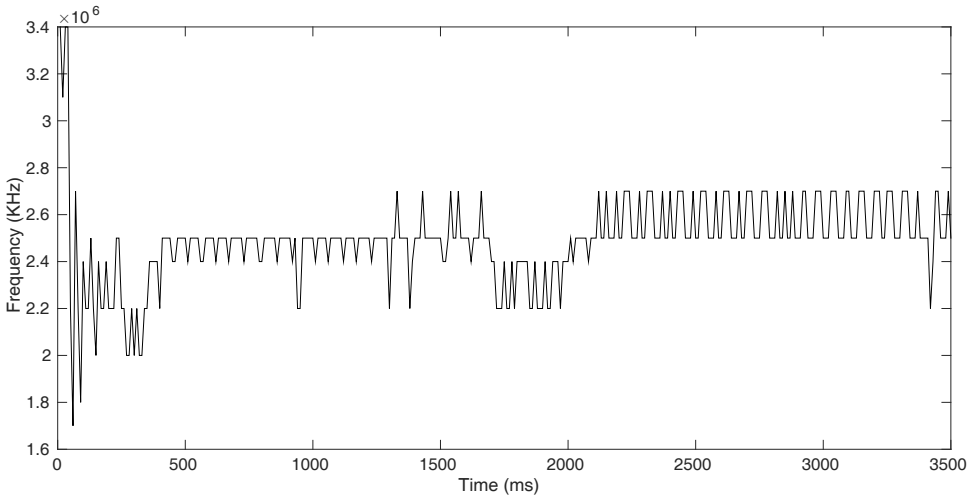

Figure 5.7: Triangle Count: clock frequency vs. time, target = 5 W, control cycle = 10 ms

For 20*ms* control cycles the power graph is shown in Figure 5.8, and the frequency is depicted in Figure 5.9. The power starts at 14.0043 W and after a transient period of 120 ms (or 60 control cycles) it settles in a band around 5 W. The average power in the interval [120ms, 3500ms] is 5.1280 W, which is 0.1280 MIPS over the target level of 5 W.

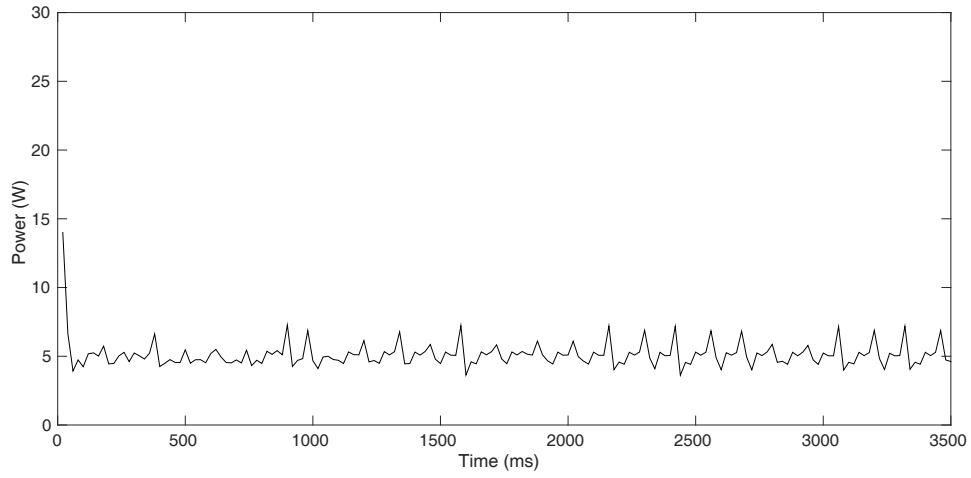

Figure 5.8: Triangle Count: power vs. time, target = 5 W, control cycle = 20 ms

For 30*ms* control cycles, the power and frequency graphs are shown in Figure 5.10 and Figure 5.11, respectively. The power starts at the value of 14.8422 W, and after a transient period of 150 ms (or 50 control cycles), it settles in a band around 5 W. Its average in the

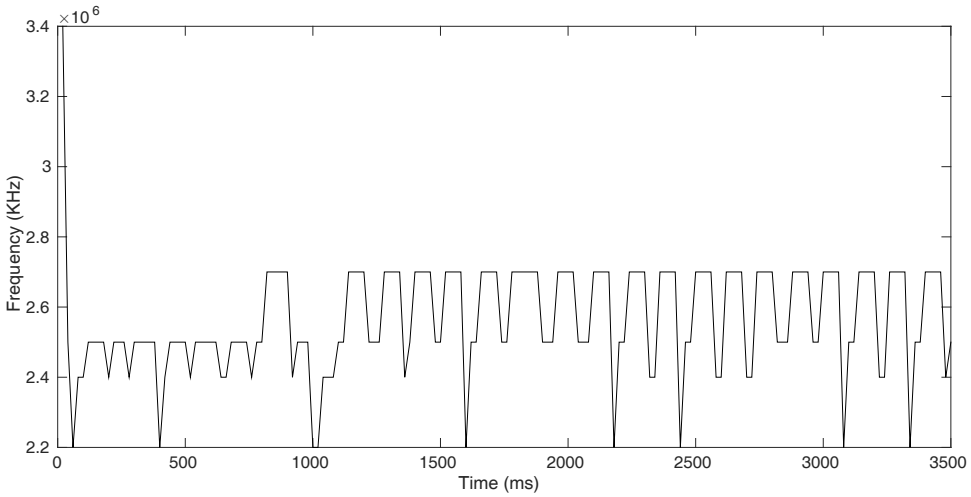

Figure 5.9: Triangle Count: clock frequency vs. time, target = 5 W, control cycle = 20 ms

interval [150ms, 3400ms] is 5.1116 W, which is 0.1116 MIPS more than the target level of 5 W.

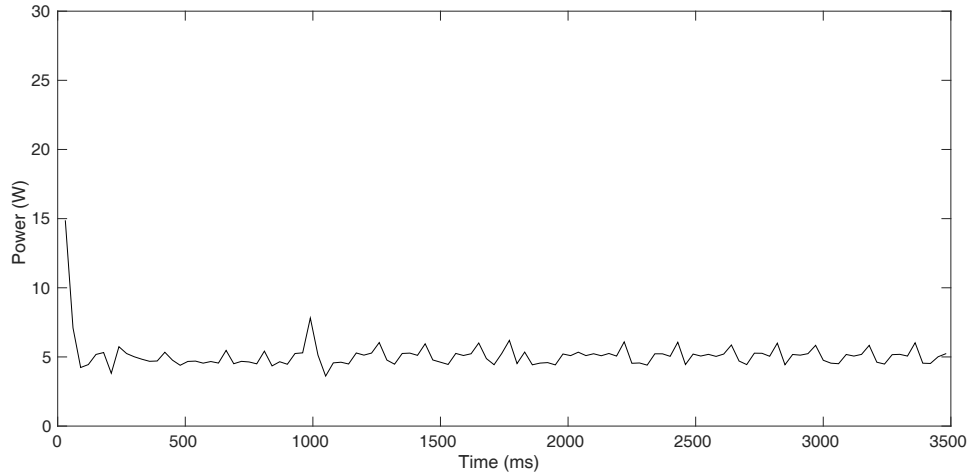

Figure 5.10: Triangle Count: power vs. time, target = 5 W, control cycle = 30 ms

Compared to 30*ms* control cycle, 10*ms* shows more oscillations. But the power variation range is the same for the three tested control cycles. The reason for that is for a smaller control cycles, the regulation is conducted more frequently resulting in more frequent operating frequency changes.

The settling time and error for Triangle Count with control cycle 10*ms*, 20*ms*, 30*ms* are

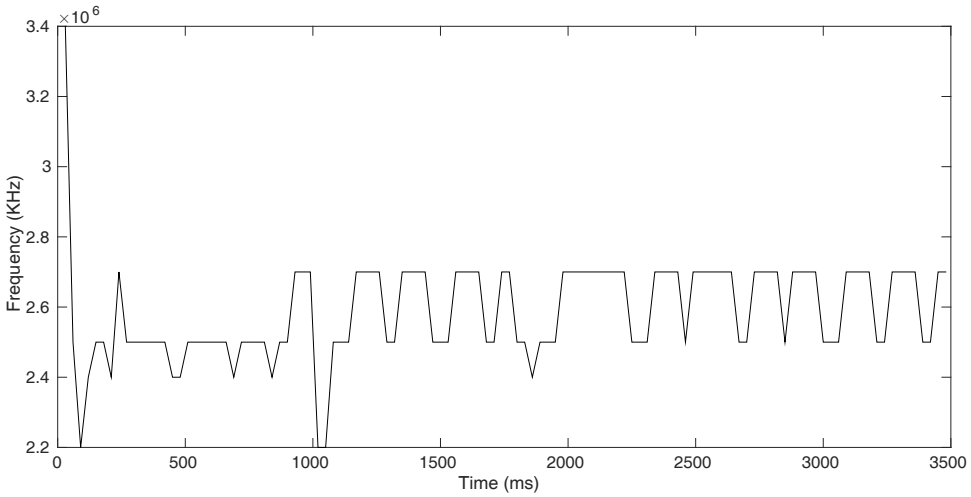

Figure 5.11: Triangle Count: clock frequency vs. time, target = 5 W, control cycle = 30 ms

shown in Table 5.2 respectively. For longer control cycle, the tracking error and the settling time generally increases. However, the fluctuation decreases as we can see in the power figures. Triangle Count has less settling time and error compared to Barnes. That indicates our power regulator can achieve the same good tracking, or even better in cloud computing applications as general industry standard benchmarks. Power and energy consumption are major concerns in data centers. Therefore, Triangle Count as a data center application is designed with improved power distribution balance.

Table 5.2: Triangle Count: average power at different control cycles

Control Cycle (ms)	10	20	30
Error (W)	0.0053	0.1280	0.1116
Settling Time (ms)	80	120	150

5.3 Concluding Remarks

In this chapter we proposed an on-line adaptive gain regulator that can precisely control the power of multi-core processors to the desired set point under various program workload. A real time power model is developed to demonstrate runtime frequency - power relationship for control accuracy and system stability. The regulator has the form of an integrator with

adjustable gain, designed for effective regulation. The gain is adjusted in real time by simple computations in the feedback loop system. Furthermore, the regulation algorithm is obtain expected power tracking in the presence of system uncertainties and computing errors in the loop. We implemented the regulator in an Intel Haswell processor in order to test it on various industry benchmarks. Due to the lack of adequate models for power evaluation of these systems, we performed a system identification algorithm that is executed in real time. We described the main technical challenges associated with implementations of the regulator. Results of the experiments are presented and discussed in detail, and they exhibit fast and effective convergence.

CHAPTER 6

POWER EFFICIENCY OPTIMIZATION FOR MULTICORE PROCESSORS

This chapter addresses the problem of optimizing power efficiency for cores as well as for single voltage island processors. In modern computing systems from edge devices to data centers, power consumption has been steadily increasing due to new technology trends as well as emerging data intensive applications. Exponential growth in data sets requires exponential growth in compute capacity but with limited growth in per processor power capacity. As a consequence, this growth in computational demand must be met by increased power efficiency of processor cores. Modern data centers use general purpose multi-core processors that are equipped with several power savings features including dynamic voltage and frequency scaling (DVFS). This chapter addresses the challenge of improving the power efficiency of those general purpose multi-core processors. In particular, we are motivated to develop an approach that is application independent. Towards this end, this chapter presents a new DVFS controller for optimizing the power efficiency of multi-core processors. The optimization controller is associated with each core or each voltage island in a multi-core processor and makes the voltage island operate at the most efficient power state with minimal compromises in performance as measured by instruction throughput.

The main contributions in this chapter are:

- A core-level adaptive gain feedback controller for increasing performance and decreasing power consumption of cores in multicore processors using stochastic approximation.

- A processor-level optimization controller for multicore processors to balance throughput and power for optimizing power efficiency of a voltage island shared by one or more cores.

- The optimization controller is implemented as a user space governor for a Linux OS in a Haswell 4-core processor. The performance is evaluated against the default and alternative governors with a wide range of benchmark applications.

6.1 Core-level Power Efficiency Optimization

The goal of our on-line controller design is to select the operating frequency for each core so as to achieve the maximum power efficiency. Figure 6.1 shows the architecture of the studied processor. Each cluster is comprised of a set of homogeneous general purpose cores whose voltage-frequency can be independently controlled. Four homogeneous cores that share a last level cache are grouped as a single cluster. Other components such as the GPU, memory controller, and network interface reside in distinct voltage islands. Each core is envisioned to have a distinct optimization controller and therefore we note that our approach is applicable to other server processor architectures with different organizations of cores. The optimization controllers of multiple cores operate independently and coordinated control among different voltage islands is not considered in this section. The optimization controller is invoked periodically at a fixed time interval referred to as the control interval to set the operating voltage and frequency for the next control interval. Each optimization controller is comprised of 3 components - i) a performance monitor which measures the collective throughput of the controlled core, ii) implementation of the optimization algorithm to compute the operating frequency for the next control interval, and iii) a power monitor which measures the collective power of the controlled core.

6.1.1 A Stochastic Approximation Approach

First, we need to find an objective function for our problem. Our goal is to improve throughput while reducing the power. Therefore, our objective function should maximize throughput while minimizing power. We define the objective function as Equation 6.1, where u_n, T_n and P_n are the operating frequency at control cycle c_n, throughput, and power for the

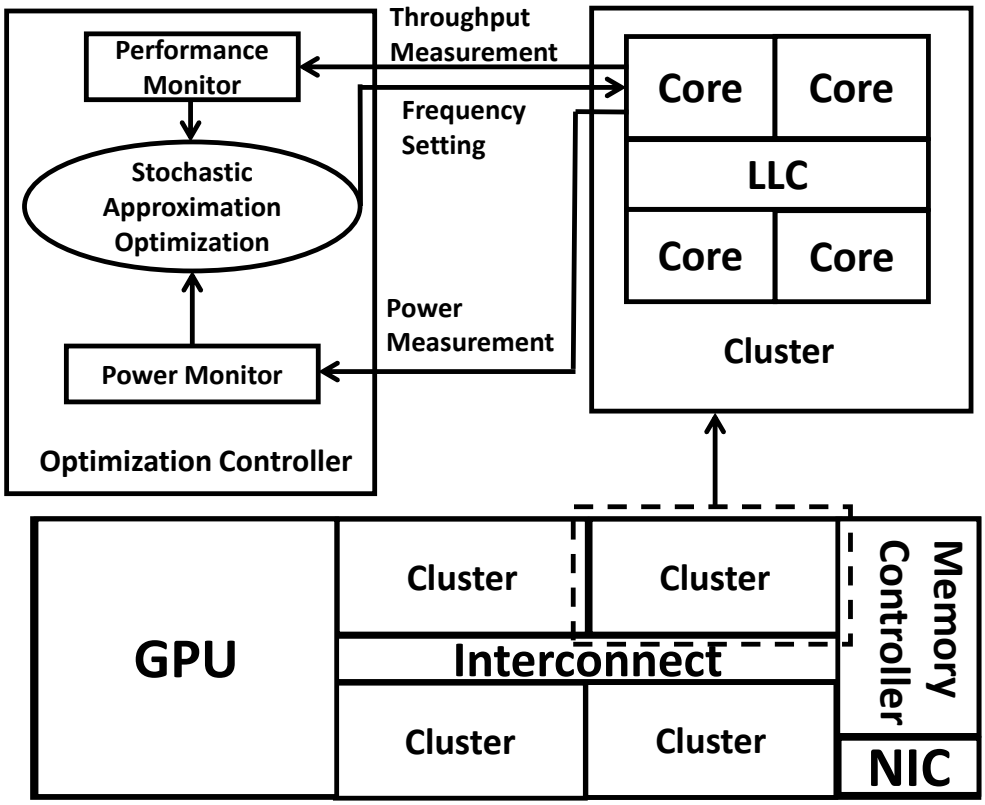

Figure 6.1: The Optimization System

controlled core at control cycle c_n.

$$\begin{aligned} &\underset{u_n}{maximize} && \frac{T_n^3}{P_n} \\ &subject\ to && u_n \in U, \quad P_n \in Powerrange \end{aligned} \tag{6.1}$$

The reason for choosing T_n^3 instead of T_n is to balance the impact of frequency on power and throughput as described in [84]. The dynamic power and frequency relationship is given by $P_{dynamic}(f,V,t) = k(t)CV^2f$ where $k(t)$ is a time-varying workload parameter representing the switching activity, V is the supply voltage, C is the capacitance, and f is the frequency. While throughput is proportional to frequency, power is proportional to the product of the square of the voltage and frequency. The proposed objective function

represents a balance of changes in power vs. throughput as a function of frequency.

The challenge in designing such an optimization algorithm is that the optimization model must be solved on-line with low computation complexity. A simple but efficient design is required. Stochastic approximation has low computing costs and high tolerance to errors[85] . Therefore, we use a stochastic approximation approachto solve the maximum problem above efficiently and accurately.

Our on-line optimization algorithm is illustrated in Figure 6.2. We start running the application with an initial frequency and collect the performance and power statistics at every control cycle. At control cycle c_n, the optimized working frequency u_n is given by,

$$u_{n+1} = u_n + \alpha_n \cdot \frac{\partial(\frac{T_n^3}{P_n})}{\partial u_n} \tag{6.2}$$

where α_n is the step size, which is generally a monotone-decreasing sequence changing with time [86]. α_n must meet two requirements: 1) $\sum_{n=1}^{\infty} \alpha_n = \infty$; and 2) $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$. We set α_n as follows to balance the response time and accuracy (see [85]).

$$\alpha_n = 0.8 \cdot \frac{1}{n^{0.6}}. \tag{6.3}$$

Hence we have

$$\frac{\partial(\frac{T_n^3}{P_n})}{\partial u_n} = \frac{1}{P_n}\Big(3P_nT_n^2\frac{\partial T_n}{\partial u_n} - T_n^3\frac{\partial P_n}{\partial u_n}\Big).$$

The quantities T_n and P_n are measured on-line. However, the terms $\frac{\partial T_n)}{\partial u_n}$ is estimated by the throughput model described in Section 4.2; and $\frac{\partial P_n}{\partial u_n}$ is estimated by the power model as described in Section 5.1.

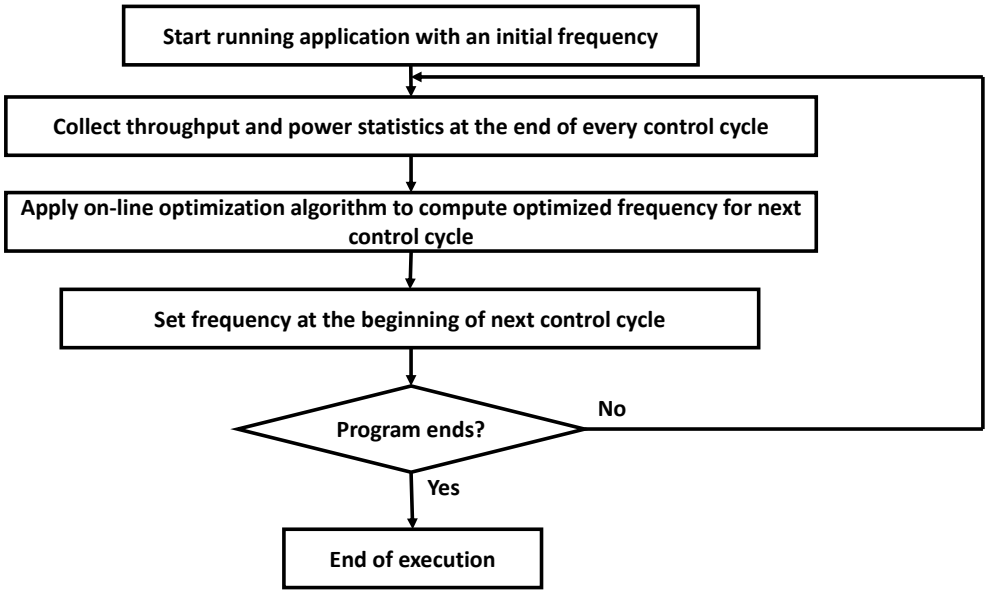

Figure 6.2: Flowchart for the On-line Optimization Algorithm

6.1.2 Experiments in a Full System Cycle Level Simulator

We use Manifold to test the proposed power efficiency optimization technique. We test our optimization controller with Splash II benchmarks: Water-ns, Barnes, Lu-c, Cholesky, Radiosity and Ocean-nc, with frequency range between $0.5GHz$ to $5GHz$. First, we execute all benchmarks with constant frequencies, which are the highest frequency $5GHz$ and the lowest frequency $0.5GHz$ in the frequency range. The average power efficiency of each benchmark with constant frequencies are described in Table 6.1 and Table 6.2.

Table 6.1: Power Efficiency for $0.5GHz$ Constant Frequency

Benchmark	Power (W)	Power Efficiency: *e27
Barnes	1.12	1.4465
Lu-c	0.78	0.3897
Radiosity	0.97	1.6616
Water-ns	0.83	1.4629
Ocean-nc	0.95	3.8907
cholesky	1.36	1.2652

Next, we run all benchmarks with the proposed design with per-core power efficiency

optimization controllers. The power efficiency results are show in table 6.3.

Power efficiency with constant frequency is much lower than those with our optimization technique. Table 6.3 also shows the average power of the tested benchmarks when using power efficiency optimization technique. Further, we also compare the optimization design with power regulation. We use the power regulator presented in Chapter 5. We use the average power in power efficiency optimization experiments shown in Table 6.3 as the target power for the regulator. In other words, we compare the power efficiency of those benchmarks with power efficiency controllers against those with power tracking controllers. The results are shown in Figure 6.3. As we can see, there is an average improvement of 38% power efficiency in the processor that implemented our optimization controller.

In order to test our design in ultra low-power system such as portable devices, we test our optimization controller with a frequency range between $0.2GHz$ to $1GHz$. The results are shown in Table 6.4. The power efficiency for power efficiency optimization and

Table 6.2: Power Efficiency for $5GHz$ Constant Frequency

Benchmark	Power (W)	Power Efficiency: *e27
Barnes	12.318	8.1942
Lu-c	6.4557	1.8747
Radiosity	8.6659	4.8383
Water-ns	6.6251	5.1096
Ocean-nc	11.1452	17.450
cholesky	9.893	17.336

Table 6.3: Power Efficiency Using The Optimization Controller With Frequency Range $0.5GHz$ to $5GHz$

Benchmark	Power Efficiency (*e27)	Average Power(W)	Average Frequency (GHz)
Barnes	49.475	5.3183	3.0480
Radiosity	37.189	7.4449	4.8201
Water-ns	68.83	7.74	4.3485
Ocean-nc	54.067	7.4631	3.7660
Cholesky	82.803	5.6527	2.8137
Lu-c	16.616	6.9929	4.7312

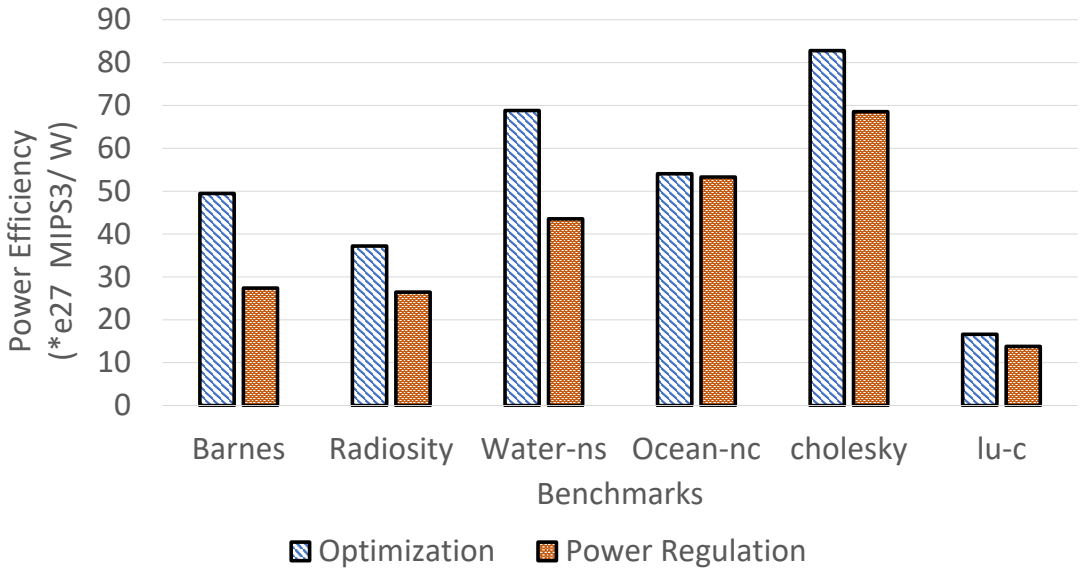

Figure 6.3: Normalized Power Efficiency (with frequency range between 0.5GHZ to 5GHz)

power regulation are shown in Figure 6.4. The average improvement of power efficiency for processor frequency range between 0.2GHz to 1GHz is 30.3%. The average power efficiency of each benchmark with constant frequency 0.2GHz is over 50% lower than those with the optimization controller as shown in Table 6.5.

Table 6.4: Power Efficiency Optimization Controller With Frequency Range From 0.2GHz to 1GHz

Benchmark	Power Efficiency (*e27)	Average Power(W)	Average Frequency (GHz)
Barnes	49.741	5.6819	3.2373
Radiosity	55.556	5.6732	4.1681
Water-ns	42.405	5.4150	4.1681
Ocean-nc	53.44	5.7023	2.0769
cholesky	69.781	5.6527	2.8973
Lu-c	17.166	6.1480	4.682

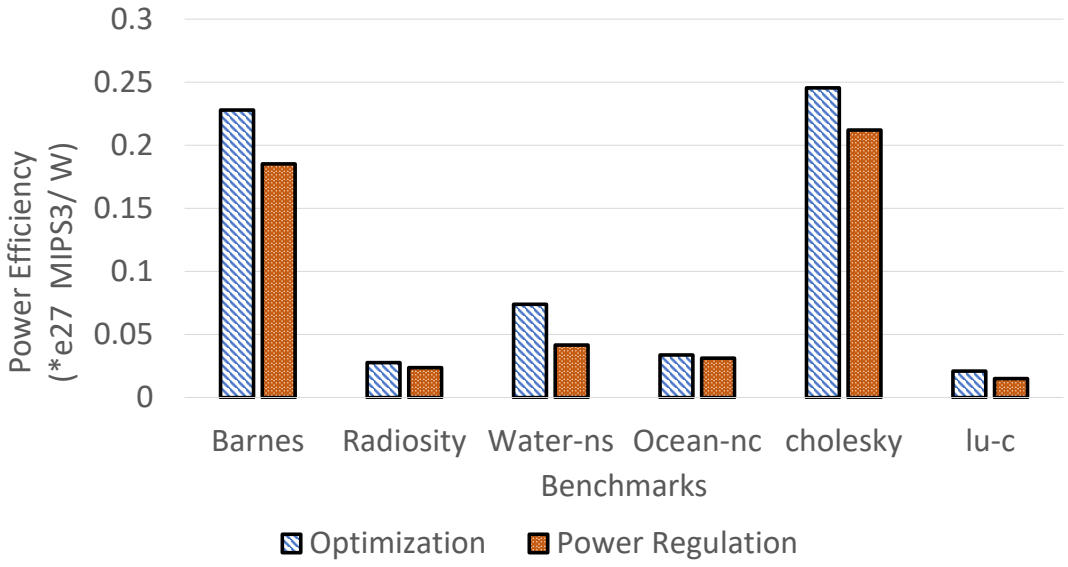

Figure 6.4: Normalized Power Efficiency (with frequency range between $0.2GHZ$ to $1GHz$)

6.2 A Processor-level Power Efficiency Optimization Controller

In this section, we implemented our power efficiency optimization controller in the Haswell processor that was introduced in Chapter 3. As explained in Chapter 3, cores on the tested processor reside in a single voltage island, where we can only set one frequency via the Userspace governor. Let's denote the average core throughput on the voltage island at control cycle c_n as T_n. T_n is calculated by measuring throughput from all four cores and dividing the total throughput by 4 to get average per-core throughout. The average per-core

Table 6.5: Power Efficiency for $0.2GHz$ Constant Frequency

Benchmark	Power (W)	Power Efficiency: *e27
Barnes	1.12	1.4465
Lu-c	0.78	0.3897
Radiosity	0.97	1.6616
Water-ns	0.83	1.4629
Ocean-nc	0.95	3.8907
cholesky	1.36	1.2652

power consumption is P_n. The technique can be extended to multiple voltage islands cases by simply assigning optimization controllers to each voltage island.

The main objective of optimization is to provide the operating frequency for solving the optimization problem in Equation 7.1. In addition, the operating frequency must be a practical frequency in the set of Frequency-sets described in 4.4. Our algorithm is executed iteratively. At control cycle n, the optimization algorithm execute the following steps.

1. *Measurement actions*: Measure core throughput and power for the voltage island by performance counters.

2. *Computation*: Compute the clock frequency based on throughput and power information collected. The frequency is computed by Equation 6.2

3. *Set frequency*: Set the new clock frequency for the voltage island. The operating frequency computed by solving the on-line optimization problem (Equation 7.1) assumes a continuous frequency setting. However, in real systems, processors have limited discrete frequency levels. Therefore, we need to map the computed frequency to a practical operating frequency in the implemented system using Table 4.1, and set the frequency for the processor in next control cycle.

Figure 6.5 summarizes the operational implementation of the power efficiency optimization design.

6.3 Implementation in a Haswell 4-core Processor

6.3.1 Comparison With Linux Governors

We compare our design with Linux governors described in Chapter 3, i.e PowerSave, Userspace, Ondemand, and Conservative [74]. We use the Userspace governor when implementing our optimization algorithm.

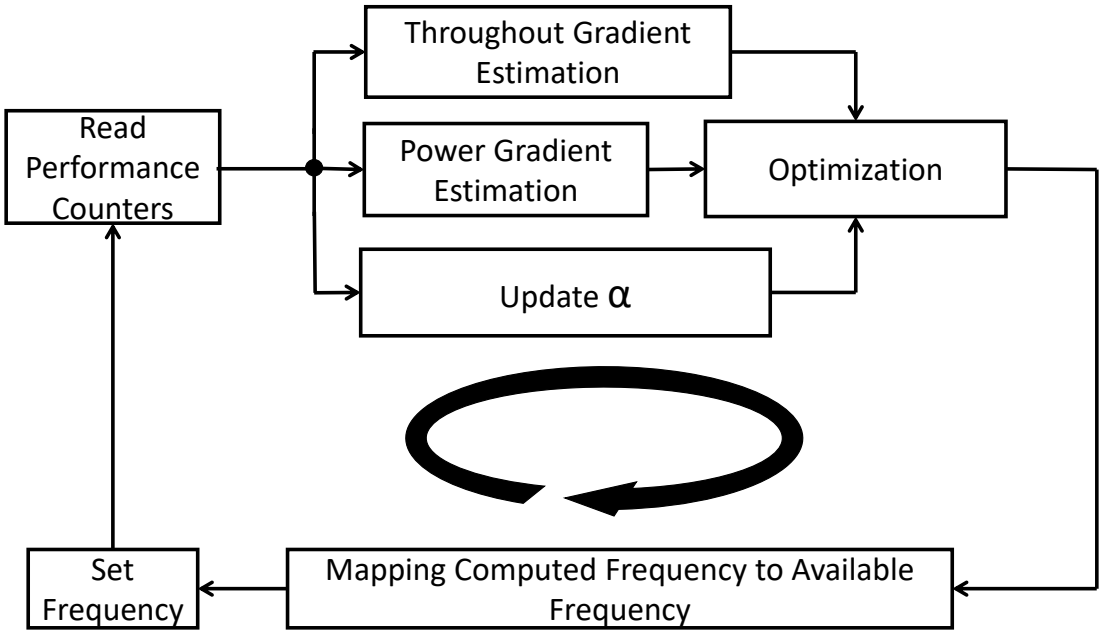

Figure 6.5: Overview of the Operation of the Power Efficiency Optimization

Figure 6.7 illustrates the execution time for several GraphBig [77] applications. As can be seen in Figure 6.7, the application run time with the Powersave governor is much more than other governors. The reason for that is the Powersave governor makes the processor always execute at the lowest frequency. This is very inefficient. Cloud computing systems require high performance to satisfy QoS requirements. So in the following analysis, we are not going to consider the Powersave governor. Since the Ondemand governor is the default governor, we compare the total energy consumption of other governors and the Userspace governor with the optimization controller to the Ondemand governor. Figure 6.8 shows the energy improvement running GraphBig benchmarks. The Conservative governor consumes less energy compared to the Ondemand governor, but the Performance governor does not gain such good energy saving. Our design consumes provides competitive energy savings in those benchmarks.

In addition to energy and power, performance is also a major concern in modern data centers. Hence, we need to compare both power and performance among those governors. EDP (Energy Delay Product) Takes into account that one can trade higher energy for

reduced delay. Hence, we present EDP improvement against the Ondemand governor in Figure 6.6. Since the Performance governor prioritizes performance over power, its EDP presents better results than the energy metric. Our design consume less energy while the execution times remain comparable as shown in Figure 6.7. The reason for better energy saving with the optimization controller is discussed in the next subsection.

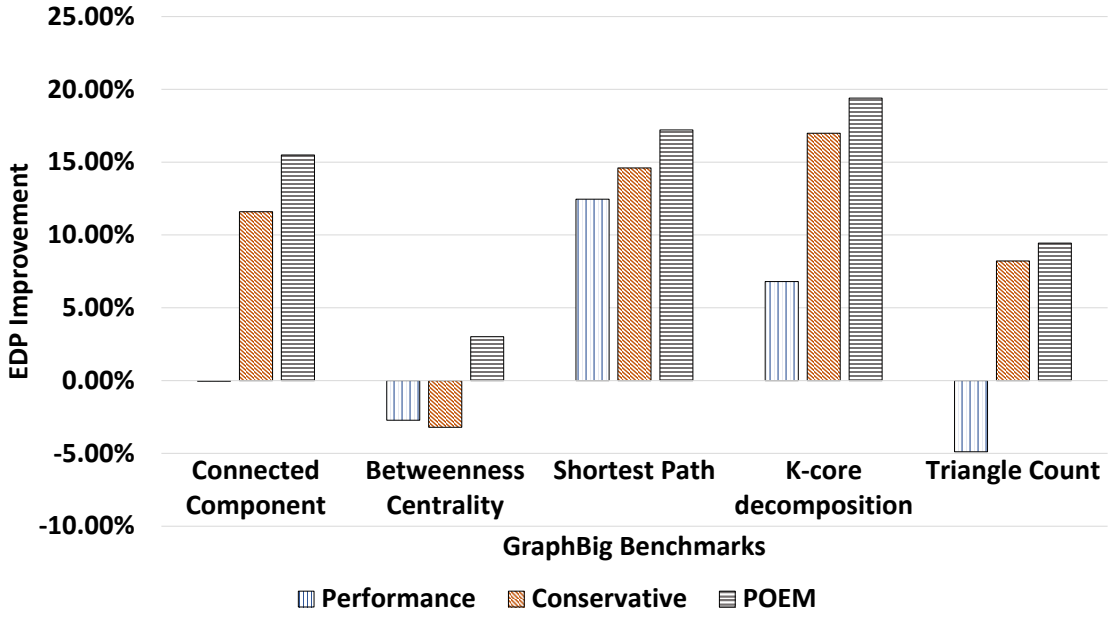

Figure 6.6: EDP Improvement Compared to Ondemand Governor

6.3.2 Experimental Results

Evaluation Metrics There are several evaluation metrics for processor designs. In particular, traditional hardware efficiency evaluation metrics are performance-per-Watt, EDP, ED2P and so one. Since the purpose of the design presented in this paper is to optimize power efficiency in multi-core processors, where both performance and power are major concerns, we use throughput-per-watt and total energy saving as two basic power efficiency metrics for comparing the optimization design with baseline in the power-performance space.

All the experimental results for the proposed optimization technique have taken over-

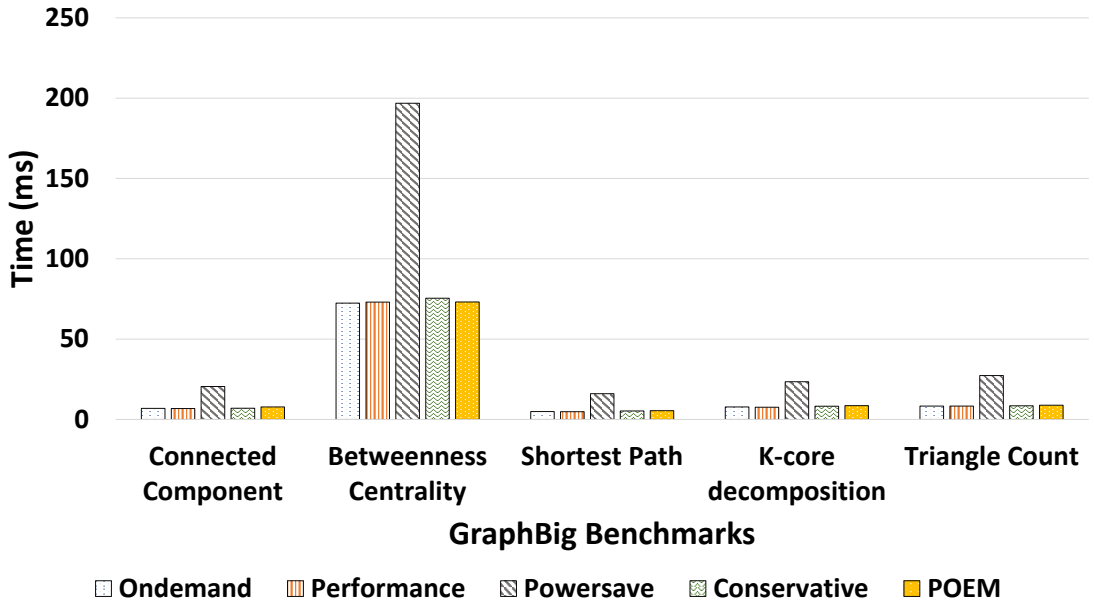

Figure 6.7: Application run time with the optimization controller and Other Linux Governors

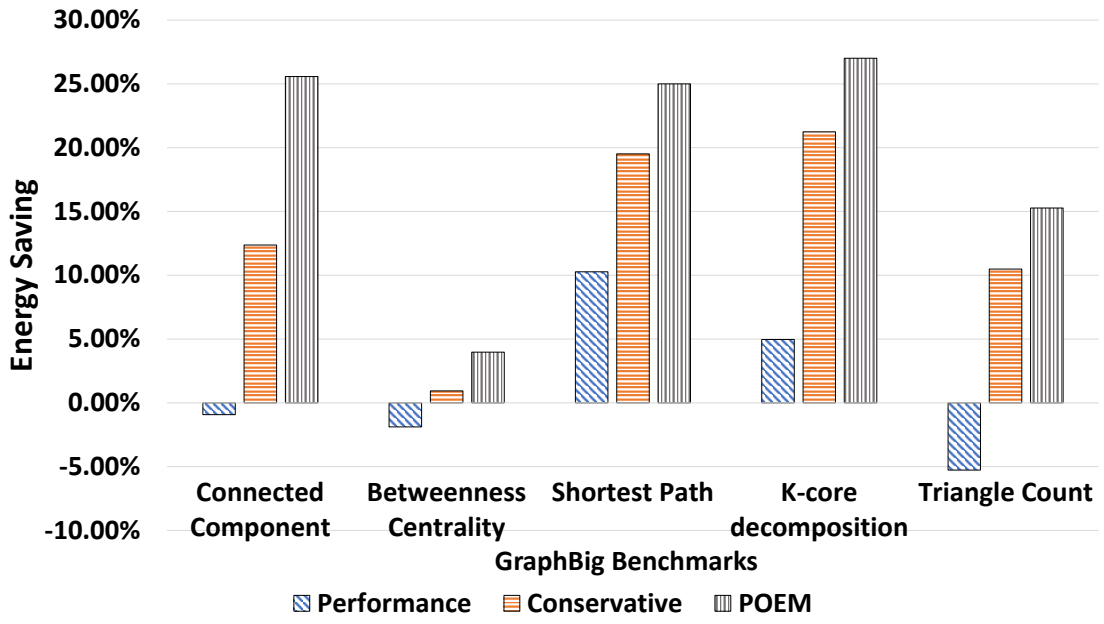

Figure 6.8: Energy Saving Compared to Ondemand Governor

head into account. The comparison baseline is to run the program with ”conservative” governor. For Splash-II benchmarks, the power efficiency (measured by throughput-per-watt), and the percentage of energy saved by running the optimization method are shown

in Figure 6.9 and Figure 6.10 respectively.

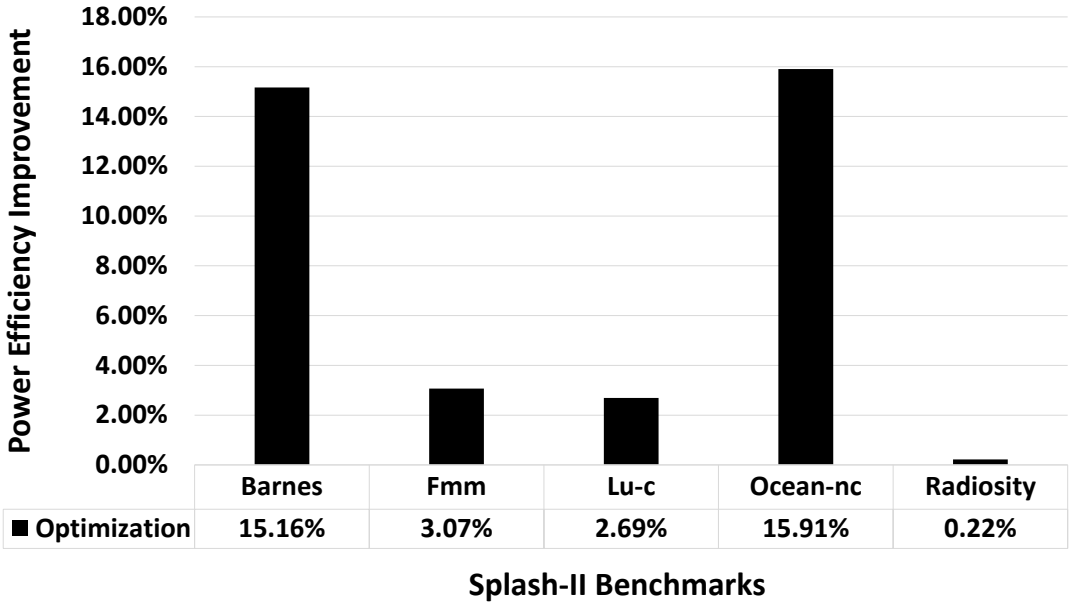

Figure 6.9: Power Efficiency (Throughput-per-Watt) Improvement for Splash-II Benchmarks

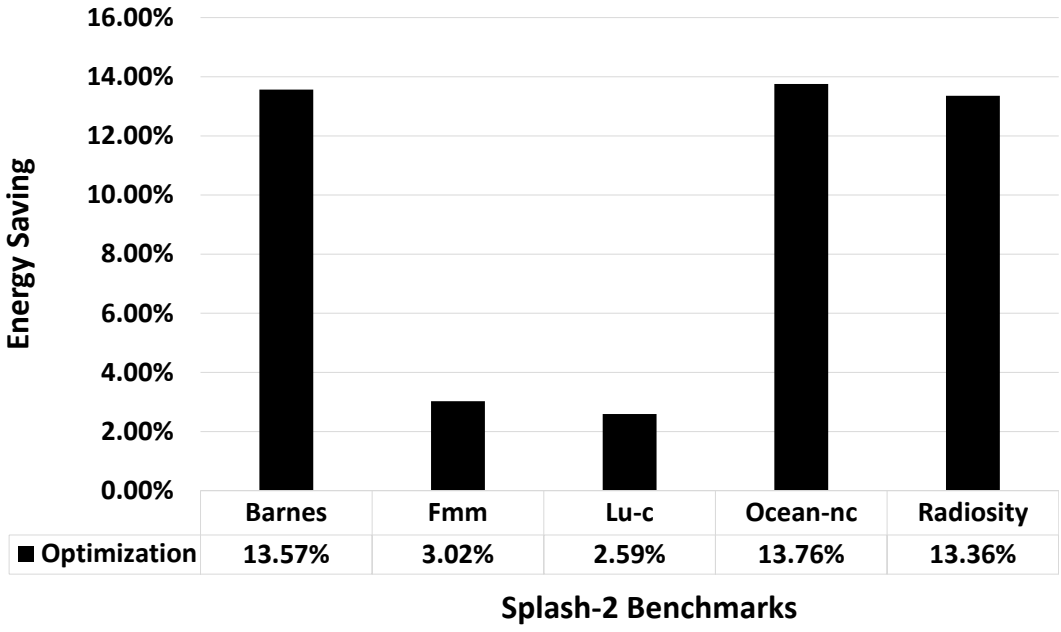

Figure 6.10: Energy Saving for Splash-II Benchmark

The power efficiency is improved up to 15.16%; while the total energy is saved up

to 13.76%. Radiosity has little throughput-per-watt improvement. That is because of the prediction error. The workload phase changes rapidly throughout the program run. However, the optimization algorithm determines operating clock frequency based on workload performance information in the previous control cycle.

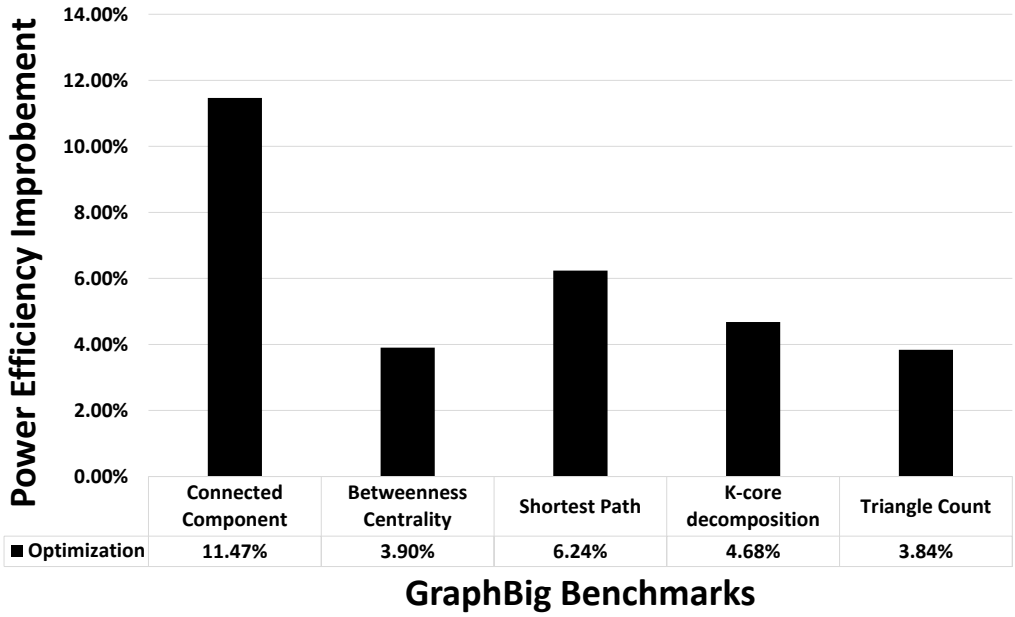

Figure 6.11: Power Efficiency (Throughput-per-Watt) Improvement for GraphBig Benchmark

For GraphBig benchmarks, the power efficiency (measured by throughput-per-watt), and the percentage of energy saved by running the optimization method are shown in Figure 6.11 and Figure 6.12 respectively. The power efficiency is improved up to 11.47%; while the total energy is saved up to 15.07%. To understand the factors contributing to power efficiency improvement in our design, we plot the throughput and power at the same figure when running the benchmark under optimization design (see Figure 6.15) and the conservative governor, which is the baseline (see Figure 6.16). The power variation appears different trend in the baseline case and optimization design. The optimization design reduces power consumption when the throughput is low, where the workload is mostly memory bounded. Because in memory phase the throughput mostly depends on memory

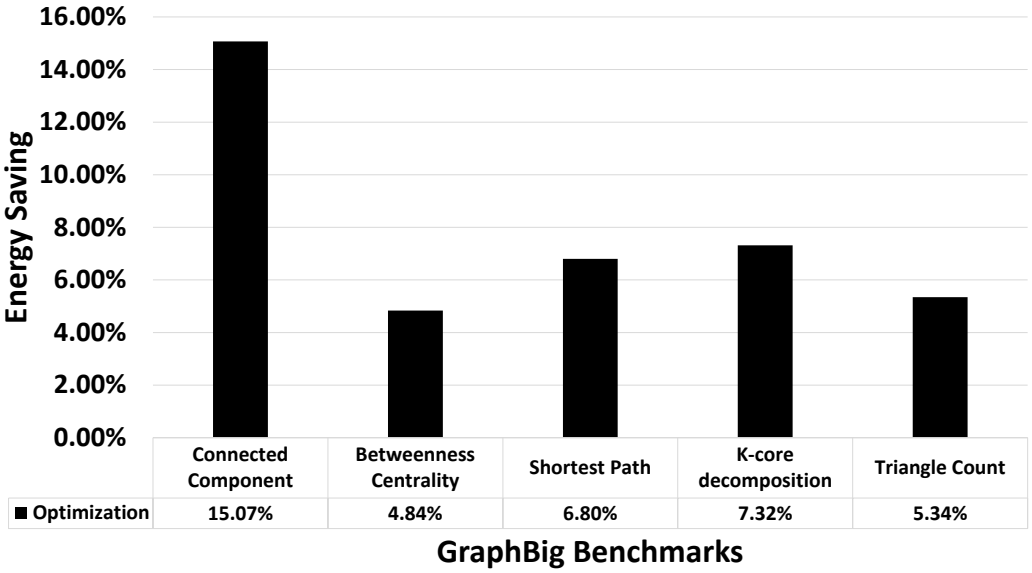

Figure 6.12: Energy Saving for GraphBig Benchmarks

execution, decreasing CPU power usage leads to little performance loss. Hence, decreasing frequency saves power and energy with marginal performance loss. The optimization design consumes more power during high throughout part of execution, where the workload is computation bounded. Increasing CPU power usage will lead to performance improvement for computation application. Hence, the optimization design gain better throughput. The throughput and power for baseline execution and optimization execution of Connected Component benchmark are shown in Figure 6.13 and Figure 6.14 respectively.

In conclusion, in comparison to the conservative governor, the optimization design reduces more power consumption during memory intensive applications, and increases more power during computation intensive applications. Therefore, the overall throughput is improved while power and energy consumption is reduced.

Similar to section 6.1.2, we also compare the power efficiency optimization design with the power regulation design. We use the power regulator described in Chapter 5 to achieve power tracking. The results for Graphbig benchmarks are shown in Figure 6.17 and Figure 6.18, and those for Splash-II benchmarks are shown in Figure 6.19 and Figure 6.19.

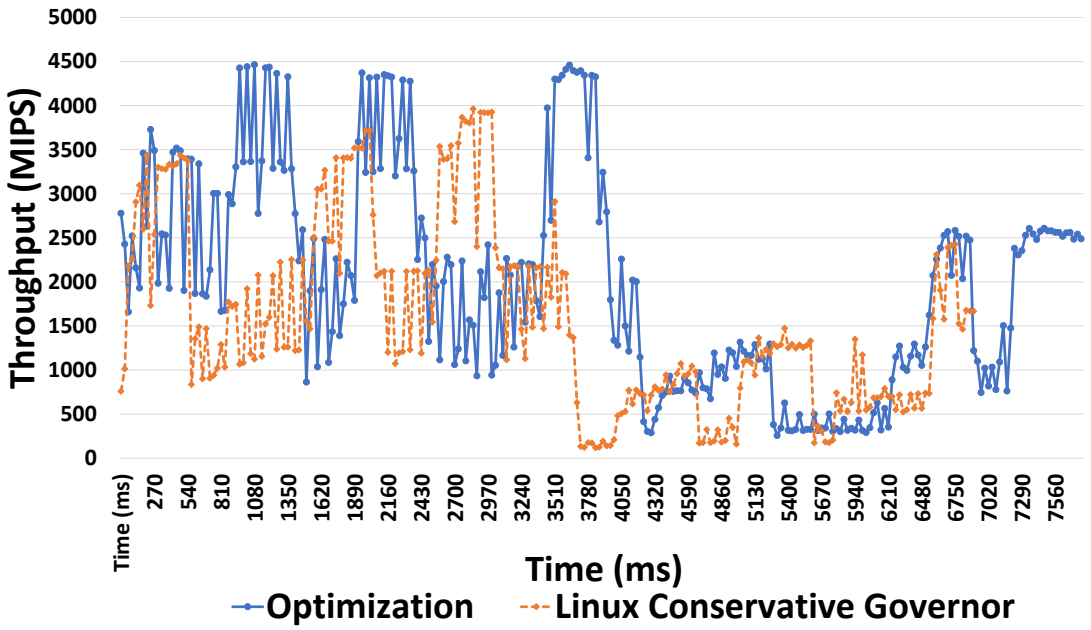

Figure 6.13: Throughput for ”Connected Component” Using the Optimization Controller and the Conservative Governor

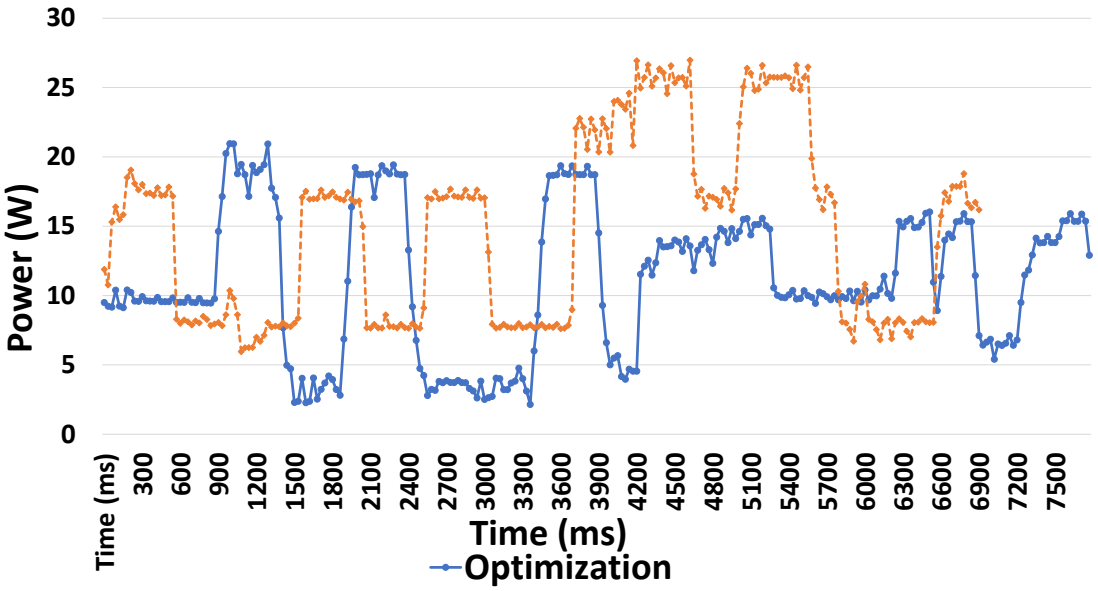

Figure 6.14: Power for ”Connected Component” Using the Optimization Controller and the Conservative Governor

Scalability analysis: Our design is a decentralized design. Each voltage island is assigned an optimization agent. Hence, it is possible to extend the proposed design to large

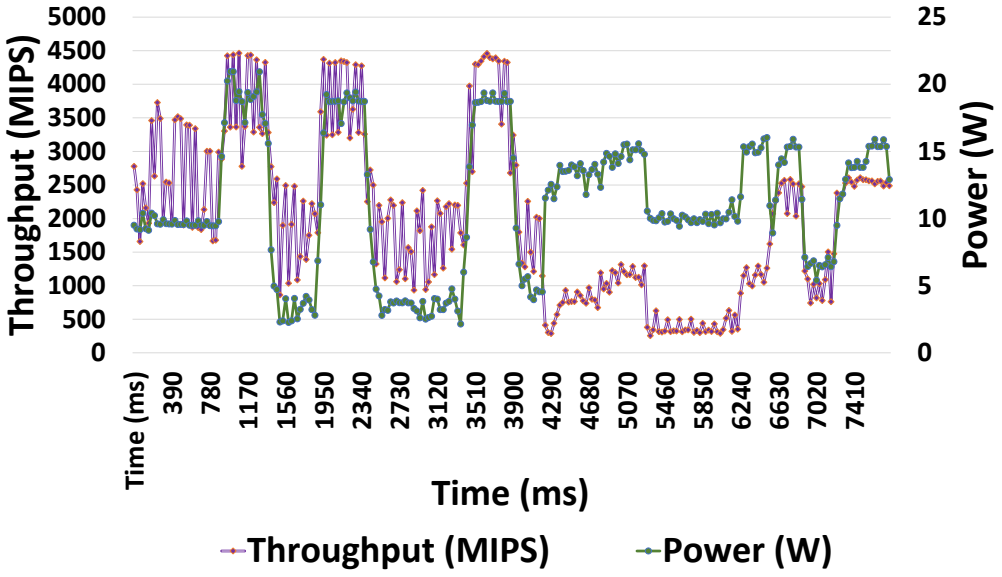

Figure 6.15: Throughput and Power for ”Connected Component” Using the Optimization Controller

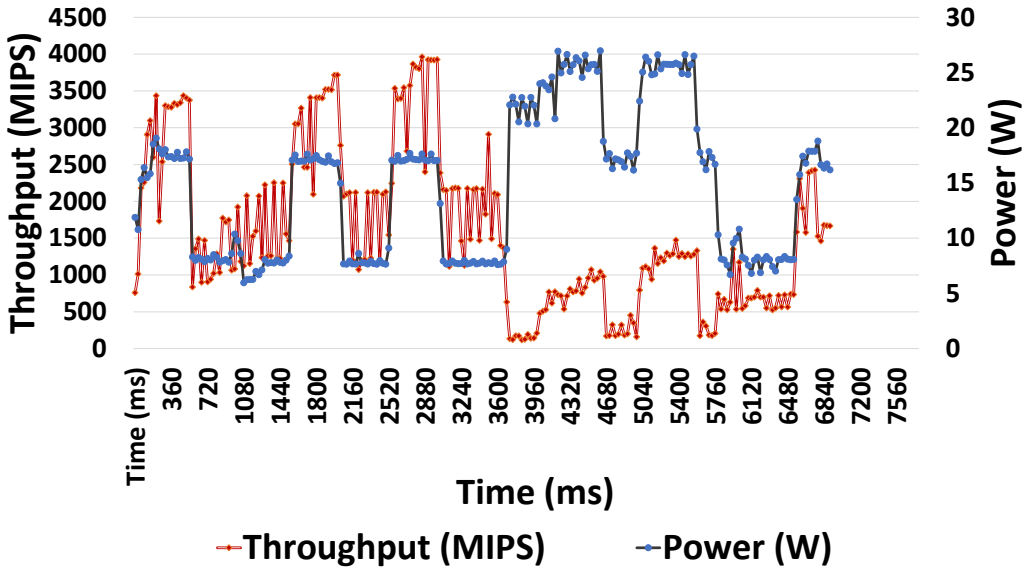

Figure 6.16: Throughput and Power for ”Connected Component” Using the Conservative Governor

data center systems, where there are large numbers of voltage islands. Implementation results from GraphBig benchmarks demonstrate that this technique can help improve power

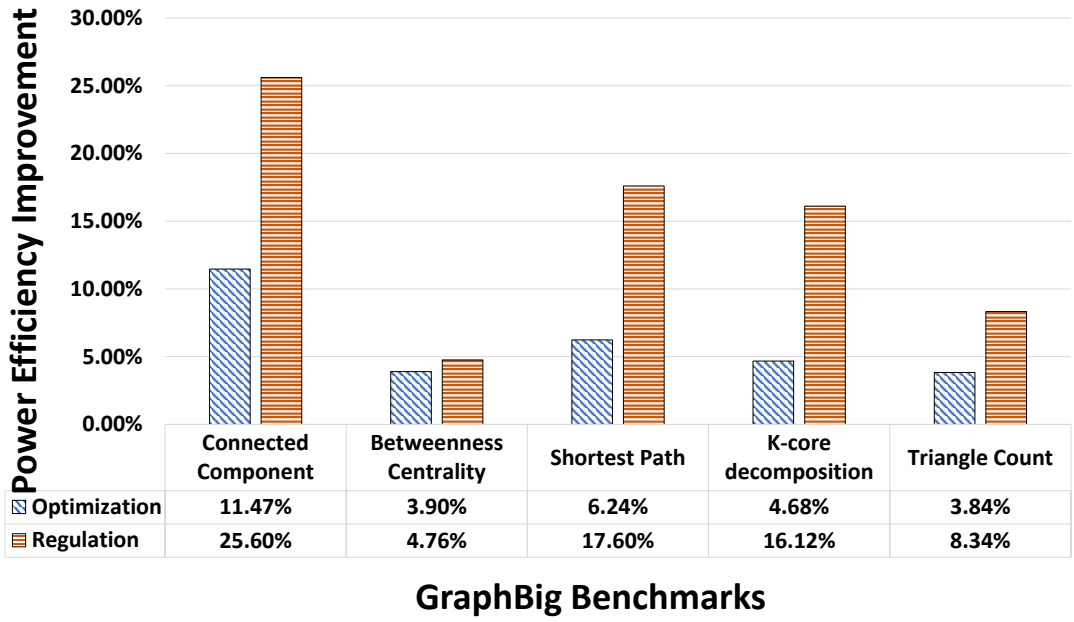

Figure 6.17: Graphbig: Power Efficiency (Throughput-per-Watt) Improvement Comparison

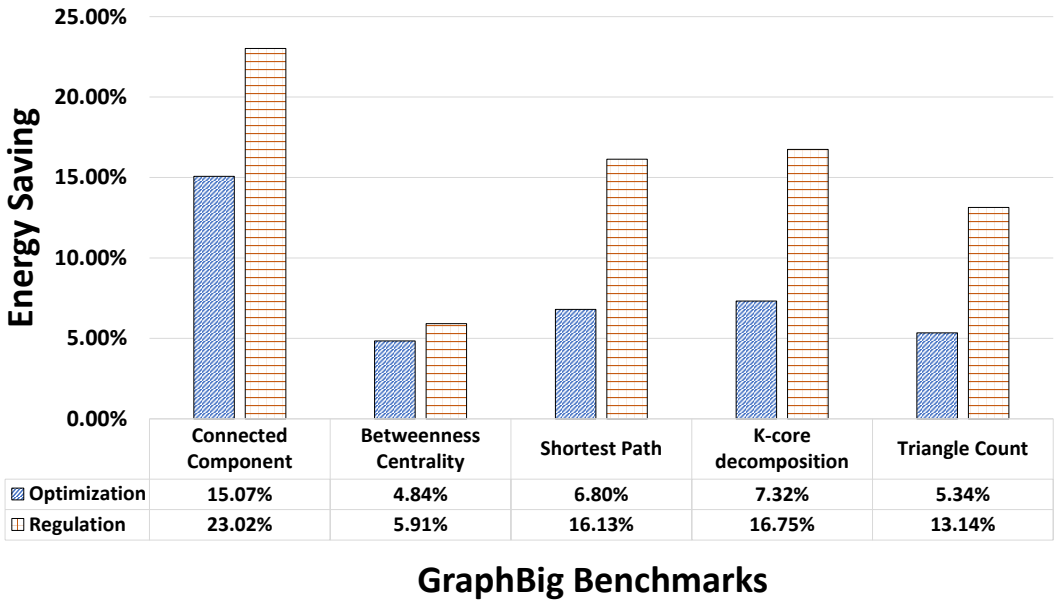

Figure 6.18: Graphbig: Energy Saving Comparison

efficiency in data center applications. The overhead in implementing the proposed optimization technique is dominated three factors: 1) optimization algorithm computation, 2) frequency setting overhead, 3) performance counters reading overhead. Those overheads

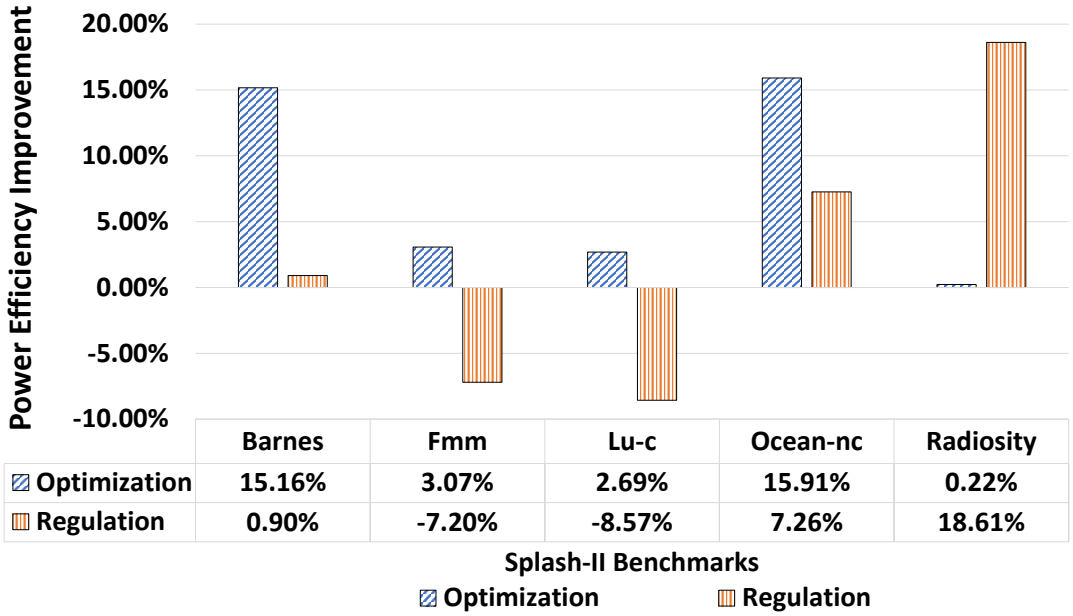

Figure 6.19: Splash-II: Power Efficiency (Throughput-per-Watt) Improvement Comparison

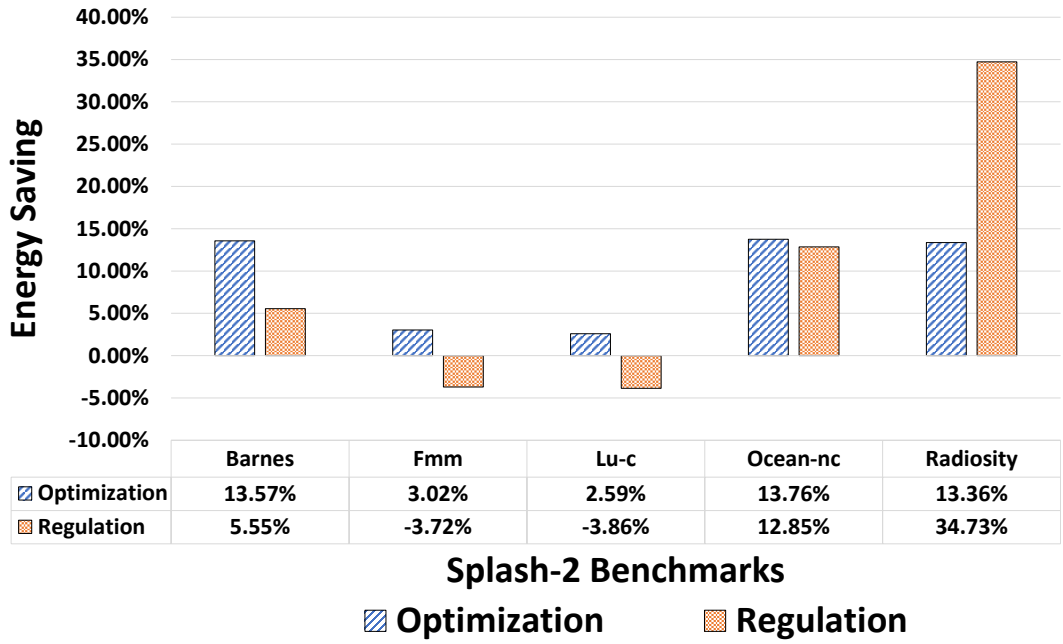

Figure 6.20: Splash-II: Energy Saving Comparison

can be controlled to desired range by properly setting control cycle time.

Error tolerance analysis : There are three main sources of errors: 1) the estimation error in the performance derivative; 2) the granularity of power model; 3) system delay.

In this we leverage on a certain robustness of stochastic approximation techniques with respect to those errors [87]. The stochastic approximation approaches have high tolerance of errors, thus is robust to errors.

6.3.3 Overhead Analysis

The presented implementation uses a control cycle of 30*ms*. The control cycle must be selected based on the computation overhead and control accuracy. In this subsection, we show the impact of control cycle duration in computation overhead and power efficiency improvement. The majority overhead comes from the computation of the optimization algorithm, and setting frequency. Since we change frequencies by setting system CPU frequency file via the Userspace governor, it takes more time and energy to complete DVFS changes compared to other Linux governors, where frequency changing is integrated with the linux kernel. We use the Conservative governor as the baseline in this section. We calculate the computation overhead by measuring the total number of instruction proceeded during benchmark execution in our design using the Userspace governor compared with the baseline. The overhead is quantified by the extra number of instructions exceeded. Figure 6.21 shows the factors leading to the optimization system overhead. To set the frequencies, we need to write system file. There is cost in Virtual File System (VFS) writing. The time and power it takes from changing the frequency to it actually taking action is unavailable to be measured. Finally, the proposed optimization algorithm also takes time and power to be computed.

To understand the influence of control cycle length on the overall system overhead, we execute the Shortest Path benchmark from GraphBig benchmark suite with a dataset size of 100K, and the control cycle duration ranging from 5*ms* to 100*ms*. We collect the total number instructions proceeded in both executions. The cumulative number of extra instructions in our design executed is expressed as a percentage of the number of instructions exceeding the baseline is illustrated in Figure 6.22. The y-axis is the percentage of

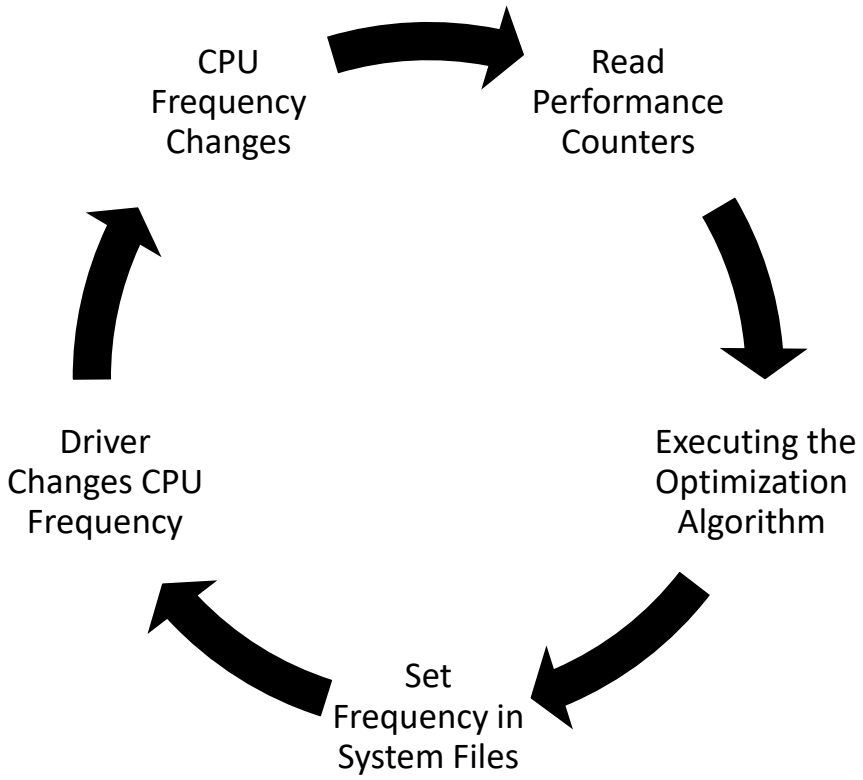

Figure 6.21: Overhead Factors

overhead compared to the baseline. As we can see, the overhead decreases as the control cycle increases. This is because the total number of control cycles decreases as the control cycle duration increases.

Next, we measure the total energy used in running ”Shortest path” (dataset size = 100K) with different control cycles for both the baseline (i.e. using ”conservative” governor), and the proposed optimization controller. The total energy saved by using our design is shown in Figure 6.23. Similar graphs were obtained from other tested benchmarks. Remember, those results have already taken the overhead into account. In other words, the presented energy saving equals the total energy consumption difference between running the program by ”Conservative” governor and with the proposed optimization technique by ”Userspace” governor. According to the results, the energy saving increases when control cycle time increases, but decreases after reaches a peak. The design consumes more energy than

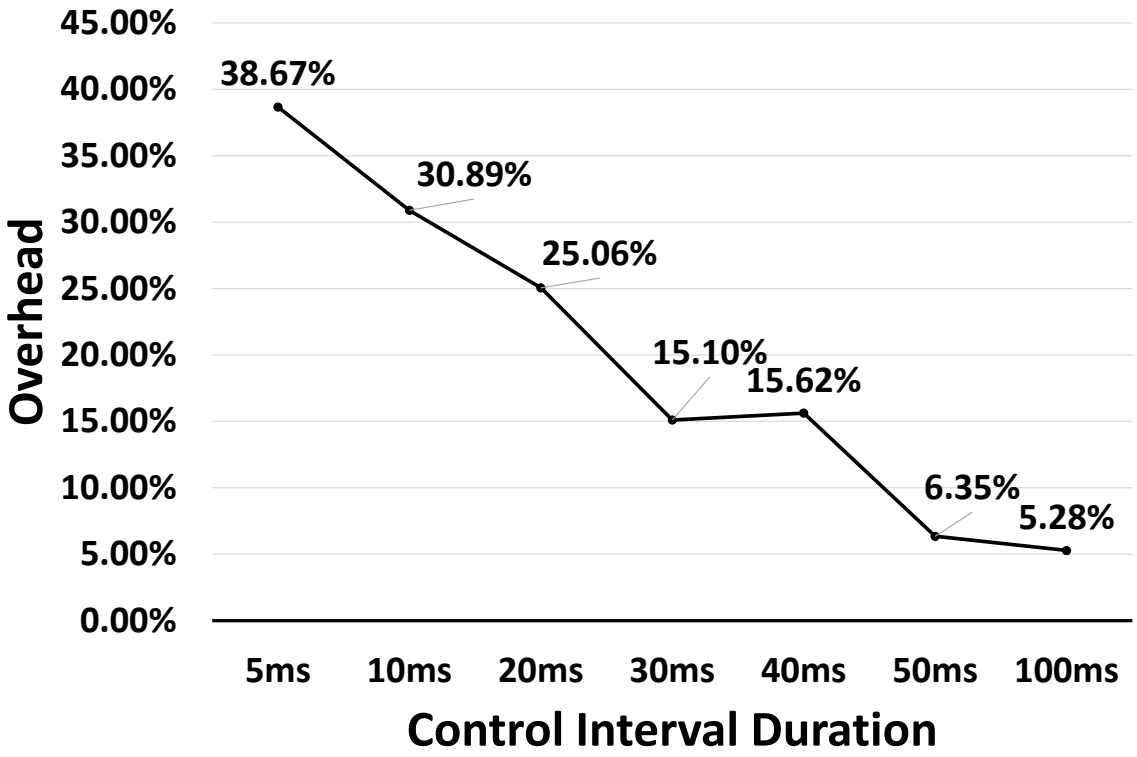

Figure 6.22: Overhead vs Control Cycle Duration

the baseline when the control cycle is too long or too short. This is because the overhead exceeds the energy saving benefit. When running the algorithm too frequently, the overhead outweighs the energy saving benefit from optimization. When the control cycle is too long, the algorithm cannot provide enough energy saving to compensate for the overhead. The principle for picking the control cycle duration is based on maximization of energy saving and minimization of computation overhead. Therefore, we choose 30*ms* as the control cycle time in the optimization design.

6.4 Concluding Remarks

In this section, we presented a simple and efficient power efficiency optimization technique for multicore processors. The optimization controller can be independently implemented

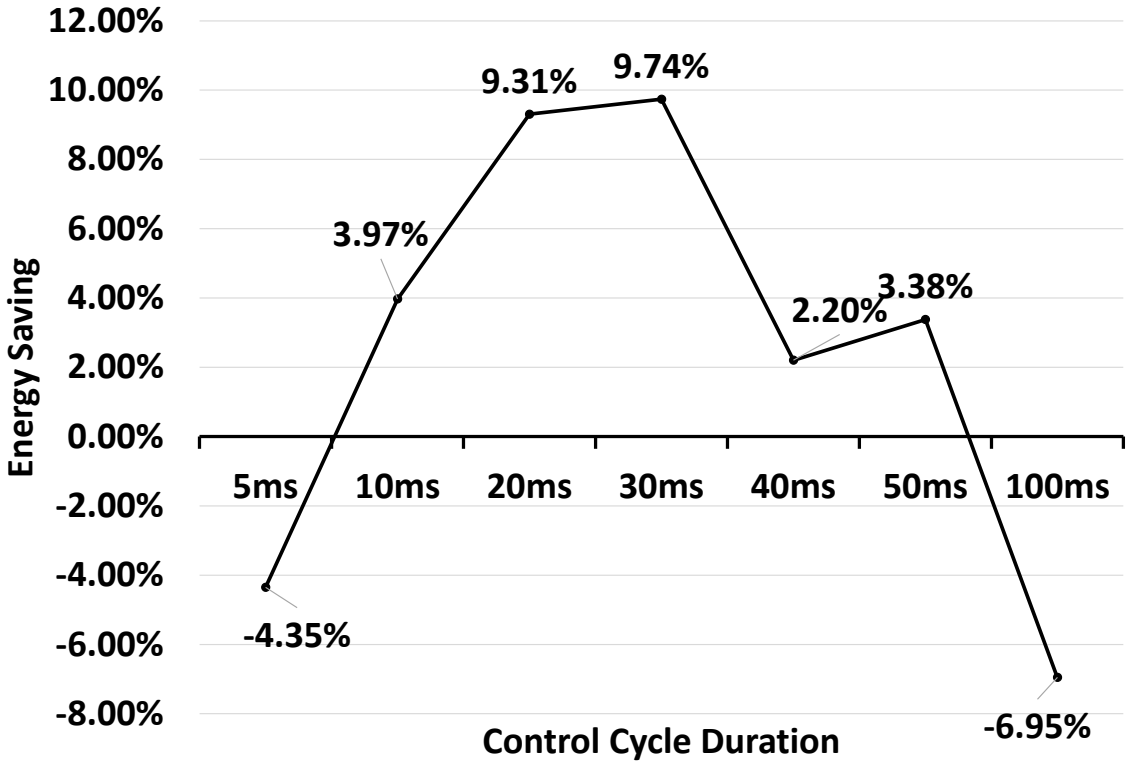

Figure 6.23: Energy Saving vs Control Cycle Duration

in a voltage domain. It is tested and simulated on Manifold simulator. We also evaluated a software implementation of the controller implemented as a Linux governor in a 4-core Intel processor using the Splash-II, Parsec, and GraphBig benchmarks. Compared to Linux Conservative governor, we show that this approach improves the power efficiency (measured as Throughout-per-Watt) up to 15.91% and the total energy saved is up to 15.07%. This combination of simplicity and robustness makes our design a good candidate for deployment in data center processors.

CHAPTER 7

POWER EFFICIENCY OPTIMIZATION UNDER POWER CAPS FOR MULTICORE PROCESSORS

This chapter addresses the problem of optimizing power efficiency and performance under power caps for multi-core processors that are composed of multiple voltage islands. Modern multi-core processors are organized into several voltage islands where each island contains one or more processing cores. Each voltage island can operate at one of several discrete power states that are defined by an operational voltage-frequency pair. Determination of the most power efficient state of a voltage island depends on many factors such as frequency, voltage, instruction stream characteristics, temperature, power and application behaviors. Our goal is to develop an on-line solution that is applicable to various applications, especially those that exhibit irregular memory reference patterns, low arithmetic density, and are often memory bound. This is challenging because the effect of operating frequencies of each voltage island on power efficiency of the processor is indirectly affected by many factors, including workload variation and the memory system behavior. According to the trade off between throughput and power in processors, two intuitive methods to improve power efficiency are: increasing performance under power caps, and decreasing power consumption under fixed throughput. In this work, we use the first method. Our objective is to improve the performance of processors by developing a controller that can improve power efficiency effectively without violating power budgets. The controller operates to balance throughput consequences and power consequences of power state transitions to obtain improved power efficiency with minimal compromises in instruction throughput. Furthermore, such a controller must operate on-line, have low overhead, adapt to time-varying application behaviors, and be portable in installations.

The main contributions in this chapter are:

- An on-line optimization technique for optimizing power efficiency as well as performance in a power capped processor that is composed of multiple voltage islands
- A dynamic power regulator to leverage power across multiple voltage islands in a multi-core processor.
- An evaluation of the proposed techniques with various applications.

7.1 A Power Efficiency Optimization Technique

In this section, we present an on-line optimization technique for improving power efficiency under a power budget. First we provide an overview of the optimization framework. The optimization system has a structure as shown in Figure 7.1. Modern processors consists of multiple voltage islands, and each voltage island is composed of multiple cores. Our idea is to assign a centralized optimization controller that dynamically changes the operating frequencies of voltage islands to optimize power efficiency. The centralized optimization controller is composed of 3 components: 1) power monitor, which collects the total power consumption of the four voltage islands; 2) optimization component, which provides operating frequencies for voltage islands by solving the optimization problem presented in subsection 7.1; 3) throughput monitor, which collects the throughput of all cores.

The optimization algorithm is conducted iteratively. At each iteration, namely control cycle, the optimization controller collects throughput and power information, computes the operating frequencies for each voltage island in the next control cycle, and then sets new frequencies.

The total power consumption of the processor is limited but the power for voltage islands can vary. Our design is to determine operating frequencies for each single voltage island in order to achieve best performance and power efficiency under power caps. A power cap is a power budget, which is the power limitation for the controlled processor.

There is a trade-off between throughput and power. Increasing frequency will poten-

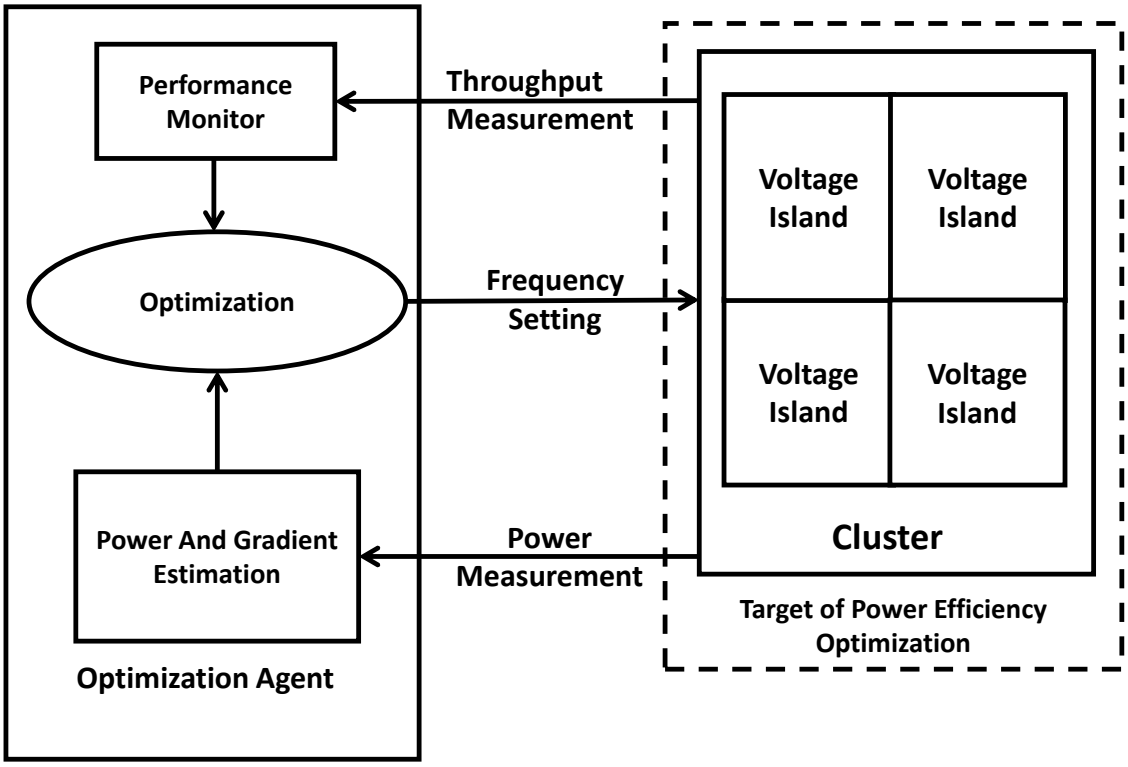

Figure 7.1: Optimization System Overview

tially improve performance, but will also increase power consumption. Therefore, we need to find the operating frequency that can leverage power for best performance. The optimization system implemented here is for four voltage islands with each voltage island composed of four cores. This technique can be extended to more voltage islands cases by simply adding more throughput and power vectors to the optimization algorithm, see section 7.1.1. The objective function is defined as maximizing the sum of the throughput of each voltage island. Furthermore, the total power consumption of the voltage islands must

be less than the power cap. Hence, we formulate the following optimization problem:

$$\begin{aligned} \underset{u_i(n)}{maximize} \quad & \sum_{i=0}^{n} Throughput_i(n) \\ subject\ to \quad & \sum_{i=0}^{n} Power_i(n) \leq PowerCap \end{aligned} \tag{7.1}$$

$n = 1,2,3\ldots$, denotes the control cycle, and $i = 0,1,2,3\ldots$ denotes the ith voltage island. In this thesis, we consider $i = 0,1,2,3$ since the studied optimization system consists of four voltage islands. $u_i(n)$ is the operating frequency of voltage island i at control cycle n. $Power_i(n)$ is the power consumption of cores and L1 cache on voltage island i at control cycle n.

Solving the optimization problem must be at low computation cost. A simple but efficiency design is required. Stochastic approximation has low computing costs and is robust to errors [86]. Therefore, we use a stochastic approximation approach to solve the maximum problem presented here.

The main objective of optimization is to provide operating frequencies for voltage islands. At each control cycle, the optimization algorithm conducts the following steps.

1. *Collecting*: Collecting throughput and power of all voltage islands, and computing total throughput and power of the processor.

2. *Computation*: Computing the clock frequencies for voltage islands based on throughput and power information collected. Details will be presented in section 7.1.1.

3. *Update frequency*: Setting the clock frequency for each voltage island.

The control cycle time must be selected based on the computation overhead and control accuracy. A small control cycle provides operating frequencies that are sufficiently close to the optimal solution of Equation 7.1, but will result in larger computation overhead since

we compute the algorithm very frequently. In this paper we choose control cycle time as $0.1ms$ to balance the speed of convergence and optimization accuracy.

7.1.1 Computing Operating Frequencies

Equation 7.1 is an optimization problem has boundary conditions. In addition to the Power Cap, we also set a power lower limit (PLL), which is usually around 10% lower than Power Budget. We divide the power consumption behavior over time into three regions, 1) less than PLL, 2) between PLL and Power Cap, 3) Over Power Cap, as illustrated in figure 7.2.

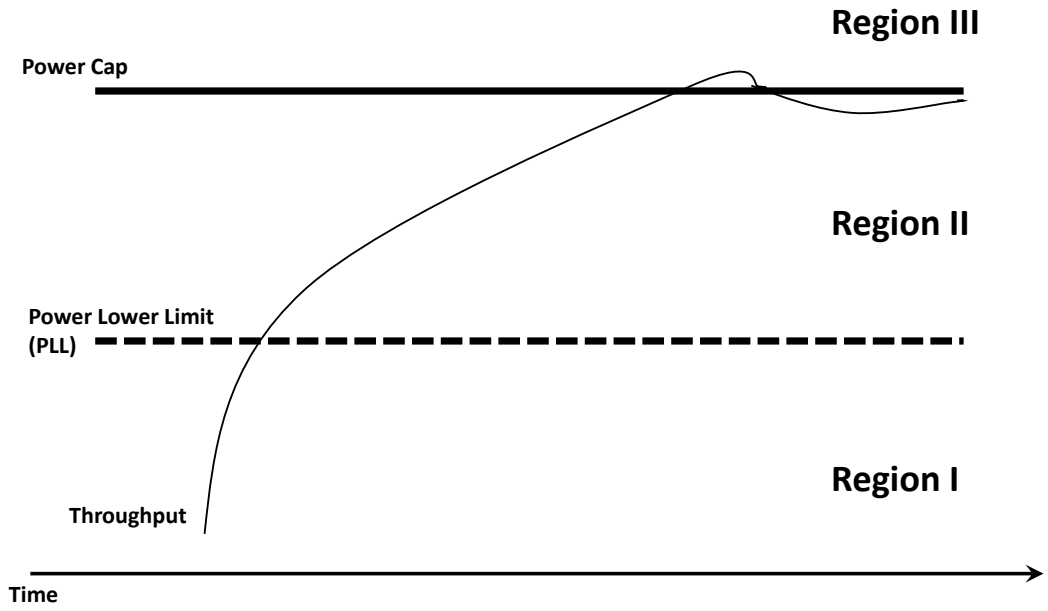

Figure 7.2: Optimization Regions

Let's define the throughput of the processor at control cycle n as T_n, which is a four dimension vector with each dimension representing one voltage island. $t_i(n)$ is the total throughput of the four cores residing on voltage island i at control cycle n.

$$T(n) = \begin{bmatrix} t_1(n) & t_2(n) & t_3(n) & t_4(n) \end{bmatrix}$$

A power vector $P(n)$, and frequency vector $U(n)$ are similarly defined. $p_i(n)$ is the power of voltage island i at control cycle n, which is the total power of all four cores and

their L1 cache residing in voltage island i, and $u_i(n)$ is the frequency of voltage island i at control cycle n.

$$P(n) = \begin{bmatrix} p_1(n) & p_2(n) & p_3(n) & p_4(n) \end{bmatrix}$$

$$U(n) = \begin{bmatrix} u_1(n) & u_2(n) & u_3(n) & u_4(n) \end{bmatrix}$$

The derivative of power with respect to frequency is denoted as δP_n, and the derivative of throughput with respect to frequency is δT_n.

$$\delta P(n) = \begin{bmatrix} \delta p_1(n) & \delta p_2(n) & \delta p_3(n) & \delta p_4(n) \end{bmatrix}$$

$$\delta T(n) = \begin{bmatrix} \delta t_1(n) & \delta t_2(n) & \delta t_3(n) & \delta t_4(n) \end{bmatrix}$$

The computation of gradients, δP_n and δT_n, will be presented in the next subsection.

Region I: $\sum_{i=0}^{3} Power_i(n) \leq PLL$

First, when the power consumption of the processor is less than PLL, $p_1(n) + p_2(n) + p_3(n) + p_4(n) \leq PLL$, our objective is to leverage frequency for maximum throughput without worrying about power. In that case, the trend of frequency change is to increase the throughput as fast as possible, which is to follow the derivative of the throughput with respect to frequency, i.e. $\delta t_i(n)$. So the operating frequencies for voltage islands at control cycle $n+1$ is:

$$U_{n+1} = U_n + \alpha_n \delta T_n. \tag{7.2}$$

where α_n is the step size, which is generally a coefficient that changes with time [85]. From [86], α_n must meet two requirements: 1) $\sum_{n=1}^{\infty} \alpha_n = \infty$; and 2) $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$. According to the basic principles in choosing α_n (see [85] for details), in this paper we pick α_n as

follows, (Remember α_n is a value that apply to all voltage islands, not a vector).

$$\alpha_n = 0.8 \cdot \frac{1}{n^{0.6}}. \tag{7.3}$$

Remember the derivative of $t_i(n)$ with respect to frequency is equal to zero for all voltage islands except voltage island i. The computation of such gradient is introduced in Chapter 3.

$$\begin{bmatrix} u_1(n+1) \\ u_2(n+1) \\ u_3(n+1) \\ u_4(n+1) \end{bmatrix} = \begin{bmatrix} u_1(n) \\ u_2(n) \\ u_3(n) \\ u_4(n) \end{bmatrix} + \alpha_n \begin{bmatrix} \delta t_1(n) \\ \delta t_2(n) \\ \delta t_3(n) \\ \delta t_4(n) \end{bmatrix} \tag{7.4}$$

Region II: $PLL < \sum_{i=0}^{3} Power_i(n) < PowerBudget$

Secondly, when the total power consumption of the processor is between PLL and Power Cap,

$PLL < p_1(n) + p_2(n) + p_3(n) + p_4(n) < PowerBudget$, our objective is to improve throughput but at the same time maintain the power consumption within Power Budget. Here we need to coordinate among voltage islands in order not to exceed the Power Cap, but at the same time leverage frequency of each single voltage island for throughput. Hence, the operating frequencies are determined by:

$$U_{n+1} = U_n + \alpha_n \delta M_n. \tag{7.5}$$

$$\begin{bmatrix} u_1(n+1) \\ u_2(n+1) \\ u_3(n+1) \\ u_4(n+1) \end{bmatrix} = \begin{bmatrix} u_1(n) \\ u_2(n) \\ u_3(n) \\ u_4(n) \end{bmatrix} + \alpha_n \begin{bmatrix} \delta m_1(n) \\ \delta m_2(n) \\ \delta m_3(n) \\ \delta m_4(n) \end{bmatrix} \tag{7.6}$$

where δM_n is a four dimension vector that is dominated by the gradient of both throughput and power.

M_n is defined as:

$$\delta M_n = \delta T - \delta T_n^T \frac{\delta P_n}{||\delta P_n||} \frac{\delta P_n}{||\delta P_n||}. \tag{7.7}$$

$$\begin{bmatrix} \delta m_1(n) \\ \delta m_2(n) \\ \delta m_3(n) \\ \delta m_4(n) \end{bmatrix} = \begin{bmatrix} \delta t_1(n) \\ \delta t_2(n) \\ \delta t_3(n) \\ \delta t_4(n) \end{bmatrix} - \begin{bmatrix} \delta t_1(n) \\ \delta t_2(n) \\ \delta t_3(n) \\ \delta t_4(n) \end{bmatrix} \cdot \frac{\begin{bmatrix} \delta p_1(n) & \delta p_2(n) & \delta p_3(n) & \delta p_4(n) \end{bmatrix}}{\left\| \delta p_1(n) \quad \delta p_2(n) \quad \delta p_3(n) \quad \delta p_4(n) \right\|} \cdot \frac{\begin{bmatrix} \delta p_1(n) & \delta p_2(n) & \delta p_3(n) & \delta p_4(n) \end{bmatrix}}{\left\| \delta p_1(n) \quad \delta p_2(n) \quad \delta p_3(n) \quad \delta p_4(n) \right\|}.$$

We use the method introduced in Chapter 4 to compute the gradient of power.

Region III: $\sum_{i=0}^{3} Power_i(n) > PowerBudget$

Finally, if the power consumption has already reached or exceeded the Power Cap, $p_1(n) + p_2(n) + p_3(n) + p_4(n) > PowerBudget$, our objective is to reduce the power consumption in order to keep the total power within Power Cap. In that case, the focus of the proposed design is to reduce power using the most expedient way even at the cost of performance loss. So the operating frequencies must follow the opposite trend of power gradient:

$$U_{n+1} = U_n - \delta P_n \tag{7.8}$$

$$\begin{bmatrix} u_1(n+1) \\ u_2(n+1) \\ u_3(n+1) \\ u_4(n+1) \end{bmatrix} = \begin{bmatrix} u_1(n) \\ u_2(n) \\ u_3(n) \\ u_4(n) \end{bmatrix} - \begin{bmatrix} \delta p_1(n) \\ \delta p_2(n) \\ \delta p_3(n) \\ \delta p_4(n) \end{bmatrix} \tag{7.9}$$

7.2 Dynamic Power Tracking

The baseline model uses the same target power for all voltage islands regardless of the application running on the cores. Performance and power is closely related to the executing applications. For example, if a processor is executing a compute intensive application, increasing the clock frequency will potentially increase the performance even though it comes with power cost. However, if a processor is executing a memory intensive application, increasing frequency may not necessarily increase the performance but still cost more in power consumption. In that case, we may reduce the frequency to save power without performance loss. Hence, we propose a dynamic power tracking technique that dynamically set power target for voltage islands based on their runtime information. The target for each regulator is changed at each control cycle.

Therefore we assign different target power to voltage islands based on their running application while keep the total power of the processors under Power Budget. More specifically, the target power is determined by the percentage of memory access in the running applications. We use bytes per op (B/O) as the metric for measuring application memory access intensity. In order to detect memory access rate of each core, we use performance counters to collect L1 cache miss instructions. Since we can measure total number of instructions executed during each control cycle, we can calculate B/O for each voltage island. A higher B/O value indicates more memory access compared to those with lower B/O value. Hence, instead of giving each voltage island the same target power, we set the dynamic target power for voltage islands that is updated at each control cycle in proportion

to B/O counters. For voltage islands on the same processor, we assign less power to those voltage islands with higher B/O and more power to those voltage islands with lower B/O value. The performance of high B/O voltage islands is dominated by memory not core. Therefore, the target power of voltage island i, $i = 0, 1, 2, 3$, i.e. $TP_i(n+1)$ at control cycle $n+1$ is given by:

$$TP_i(n+1) = \frac{1}{O_i(n)} \cdot \frac{1}{\sum_{i=0}^{3} \frac{1}{O_i(n)}} \cdot PowerBudget \tag{7.10}$$

where $O_i(n)$ is bytes-per-op of voltage island i at control cycle n.

7.2.1 Baseline Model

Since the goal is to improve performance under the condition that power must below the power cap, one intuitive solution is to make full use of the power budget but does not exceed it. In other words, processor power consumption should be maintained at the power cap. A simple power regulator presented in Chapter 4 as well as Ref [3] can achieve such desired power tracking. Hence, we use this power regulation technique as the baseline model for comparison.

The power regulator works at a voltage island level not at the processor level. So each voltage island is assigned an adaptive gain integral controller, which keeps the power consumption of the voltage island at the target power, which is the power cap in this scenario. Since there are four voltage islands in the tested processor, we implemented four power regulators. First, we set a target power for each voltage island, where the voltage islands track the target power by dynamically adjusting operating frequencies. Since there are four voltage islands residing on the processor, the target power for each voltage island is simply the Power Budget divided by four.

7.3 Experiments in a Full System Cycle Level Simulator

We implement the proposed design in Manifold. There are four clock frequency domains in the simulated processor, with a range from $0.3GHz$ to $1.5GHz$. We assume a continuous frequency settings. The operating system we use is Ubuntu 14.04. In this section, we compare the experimental results of the optimization technique, dynamic power regulation, and power regulation by running Splash-II [75], Parsec [76], and GraphBig [77] Benchmarks.

Evaluation Metrics Since the purpose of the design presented here is to optimize performance under power budget in multi-core processors, where both performance and power are major concerns, we use throughput-per-watt as the basic power efficiency metrics for comparing the optimization design in the power-performance space.

The comparison baseline is to run the program with power regulator presented in Ref [3]. We compare the experiment results from the optimization technique introduced in section 7.1.1 and the dynamic power regulator presented in section 7.2. The Power Budget for the processor is set as $15W$. We tested our design in both continuous frequency setting and discrete frequency setting. The frequency range for continuous frequency is from $0.2GHz$ to $1.5GHz$. To simulate the actual frequency settings in real computer processors, we also tested our design in discrete frequency setting. The frequency range is from $0.2GHz$ to $2.0GHz$, with 8 discrete frequency levels, (0.2GHz, 0.5GHz, 0.8GHz, 1.0GHz, 1.2GHz, 1.5GHz, 1.8GHz, 2.0GHz). The MIPS-per-Watt results are shown in Figure 7.3 and Figure 7.4 respectively. For Graphbig benchmarks, i.e. DC, TC, Kcore and Pagerank, as well as memory intensive splash-II benchmark Lu-nc, optimization technique provides best power efficiency, while for pure compute bound benchmark, i.e. Blackschores, power regulation provides best power efficiency. The optimization design shows better power efficiency improvement in data center applications and memory bounded applications. This is because it coordinates power among voltage islands based on their runtime information. Given a power budget, the optimization controller may assign more power to those voltage

islands that are executing compute applications than those voltage islands that are executing memory bounded applications. Furthermore, the optimization design can adapt the power budget for the voltage island to their executing application characteristics. However, power regulation set the power target at the beginning at the program execution and cannot exploit changes in runtime power and performance information. For compute intensive applications, the power regulation presents same or even better power efficiency compared to the other two techniques. Compute intensive applications have little variation during program execution, so tracking a pre-set power target is practical. Unlike the optimization technique, and the dynamic regulation technique has to re-set the power targets at each control cycle. Hence the power regulation technique can also benefit from the computation overhead saving.

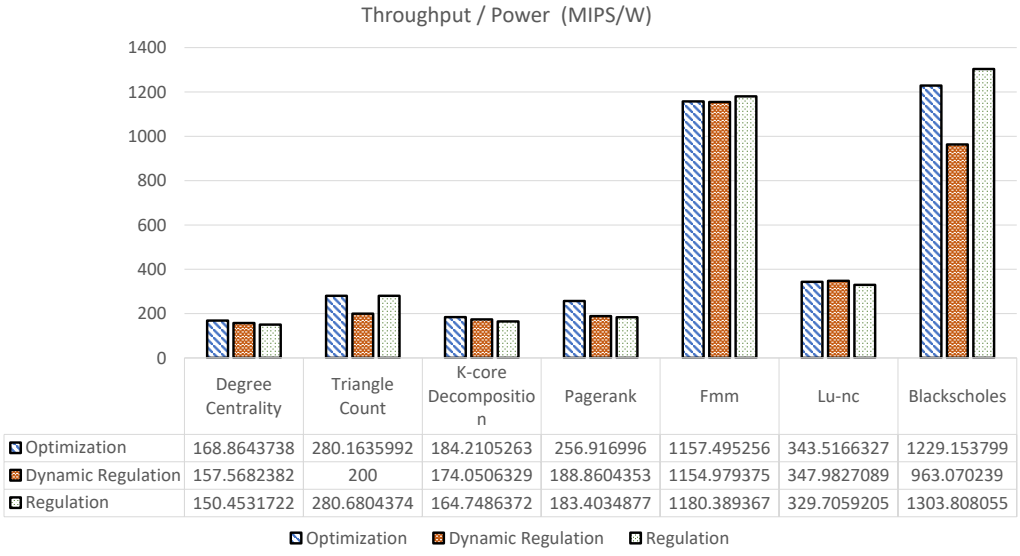

Figure 7.3: Throughput over power (MIPS per W) with 15W Power Budget

In addition to power efficiency improvement, keeping the total power consumption under Power Budget is another important issue in the proposed design. Hence, we presented the total power consumption of all three techniques in Figure 7.5 and Figure 7.6. As we can see, the optimization technique achieves the best power capping control. This is because

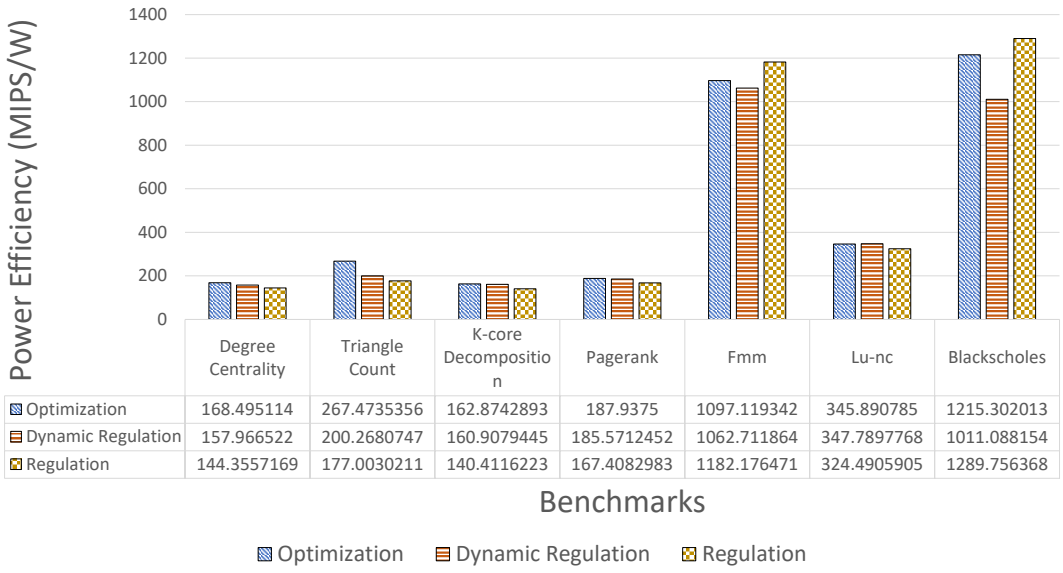

Figure 7.4: Throughput over power (MIPS per W) with 15W Power Budget under discrete frequencies

the optimization technique uses coordinated control, where power of all voltage islands are managed by the centralized controller. For power regulation and dynamic power regulation, each voltage island is managed by their own regulation controller. As a result, the regulation errors from all controllers are added together resulting in the processor power regulation errors.

In conclusion, the optimization design provides best performance and power efficiency when the workload varies widely throughout program execution. It also keeps power consumption of the processor within power budget most effectively. The dynamic power regulation presents comparatively better power efficiency in memory intensive applications than power regulation. The power regulation technique shows best power efficiency in the pure compute bound applications.

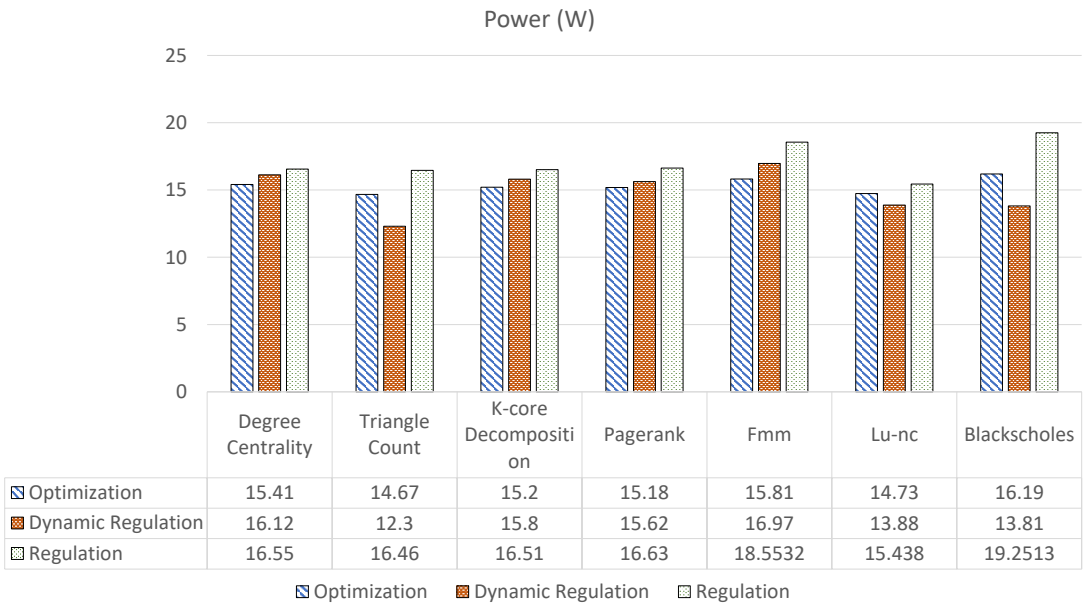

Figure 7.5: Processor power (W) with 15W Power Budget

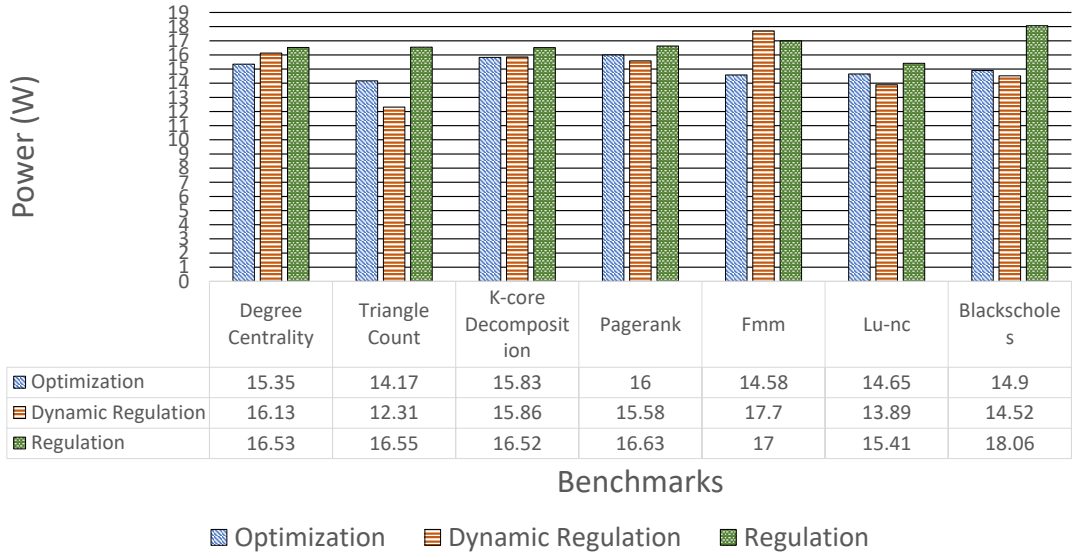

Figure 7.6: Processor power (W) with 16W Power Budget under discrete frequencies

7.4 Concluding Remarks

This chapter addresses the problem of power efficiency optimization under a power budget for multi-core processors that are composed of multiple voltage islands. We presented two

approaches and compared them with the regulation technique presented in Chapter 5. The first one is a centralized optimization controller to maximize performance within power budgets. The second one is a dynamic power regulator that re-distributes power budgets of voltage islands based on their runtime memory access information. The optimization technique is best for applications that exhibit irregular memory reference patterns, low arithmetic density, and rapid changing behaviors. The power regulation technique works best on pure compute intensive applications. Furthermore, the optimization design strictly maintains the power consumption under power caps. Hence if restricting power is the major concern in addition to power efficiency improvement, the optimization technique is the best choice.

CHAPTER 8

ENERGY EFFICIENCY OPTIMIZATION UNDER POWER BUDGETS FOR CLOUD SYSTEMS

This chapter addresses the problem of optimizing energy efficiency as well as performance for cloud computing systems. Performance variation is one of the main obstacles for efficient power usage in large scale cloud systems [88]. Bulk Synchronous Parallel (BSP) application is one of the most frequently used applications in modern cloud systems. A BSP is composed of parallel computations on each node, communication among nodes, and barrier synchronizations. The application behaviors vary significantly across nodes. Synchronization and Communication happen frequently among processors in Bulk Synchronous Parallel (BSP) applications. The nodes that arrive at the barrier first must spend idle time waiting for other nodes to arrive at the barrier. The performance is limited by the slowest node since the other nodes have to wait on barrier synchronization. This idle waiting consumes power but produces no effective throughput - thus it is a major source of inefficiency. Many other factors such as resource contention [89] and operating system (OS) interference [90] also induce the performance variation. To address the problem of efficient power usage in BSP applications, we adapt power usage to the runtime characteristics for BSP applications. We introduce a Hierarchical Power Gating and Power Shifting (HPGPS) technique that dynamically improves energy efficiency in cloud systems without interrupting runtime execution.

The main contributions in this chapter are:

- The design of a hierarchical power gating and power shifting technique (HPGPS) for improving energy efficiency and performance in large scale cloud systems.
- Evaluation of the proposed power management technique with real world application

traces.

8.1 A Hierarchical Power Gating and Power Shifting Technique

A cloud system is composed of a certain number of nodes, and each node is composed of a certain number of processors. HPGPS assumes node level power gating, which means all the processors belonging to the same node can either be all power gated, or none of them are power gated. The cloud system is assigned a power budget, and each node is also assigned a node power budget. In the beginning of execution, we distribute power budget evenly to all nodes. Hence, the node power budget is equal to the cloud power budget divided by the number of nodes.

In a barrier synchronization, nodes arriving at the barrier can not process further execution until all nodes arrive at the barrier, see Figure 8.1. This waiting consumes power but produces no performance. Hence, we can power gate those nodes that have arrived at the barrier (fast nodes), and shift the power budgets for fast nodes to other nodes that are still under execution (slow nodes). The extra power will speed up the execution of slow nodes, thus slows nodes will arrive at the barrier earlier than otherwise, see Figure 8.2. Hence the barrier waiting time is reduced. As a result, the total execution time for the program is reduced. Since the power budget remains the same, less execution time saves energy. With less time for program execution, the performance is also improved. This is the centralized power shifting and power gating technique presented in [66]. This technique works well in small data center systems. However in large cloud systems which contain thousands of nodes, this technique cannot be applied due to large delay in network communication. When the cluster size is large enough, even one time power shifting communication delay will be longer than the application execution. In order to apply power shifting and power gating technique in large scale data center systems, we developed HPGPS that can tolerate the large communication latency in cloud systems. HPGPS divides nodes in the cloud into separate groups with adaptive group size (the number of nodes in a group). A centralized

controller is used to determine group size. Each group has a group controller to conduct power gating and power shifting within the group. Figure 8.3 illustrates the hierarchical power management scheme.

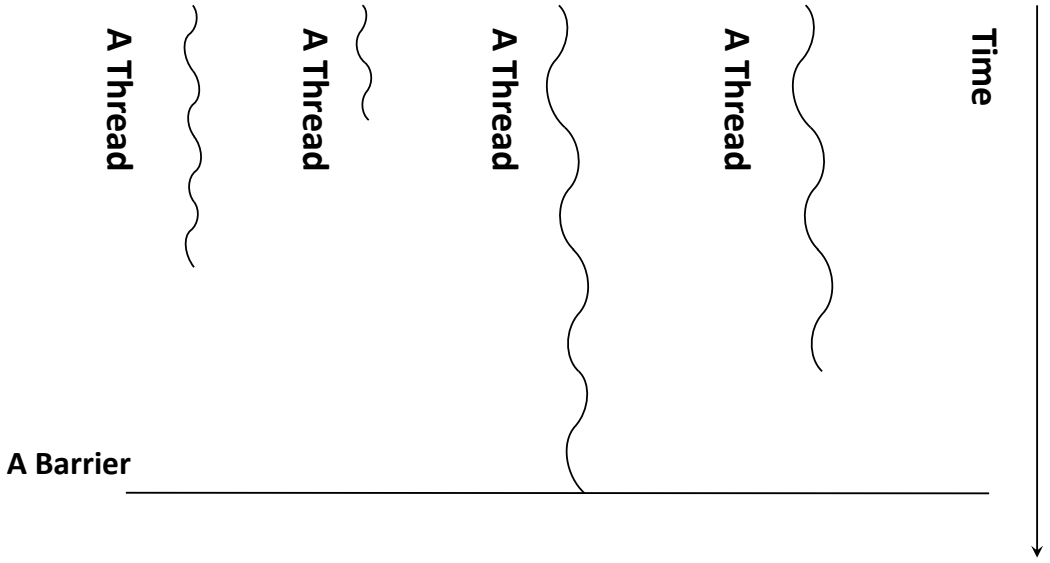

Figure 8.1: Barrier Synchronization

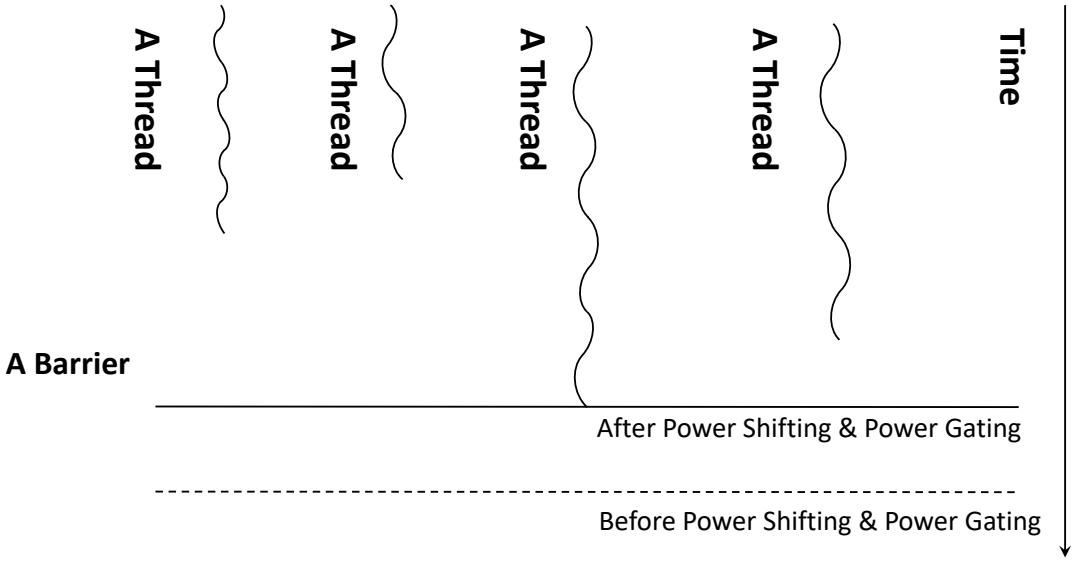

Figure 8.2: Barrier Synchronization After Power Shifting & Power Gating

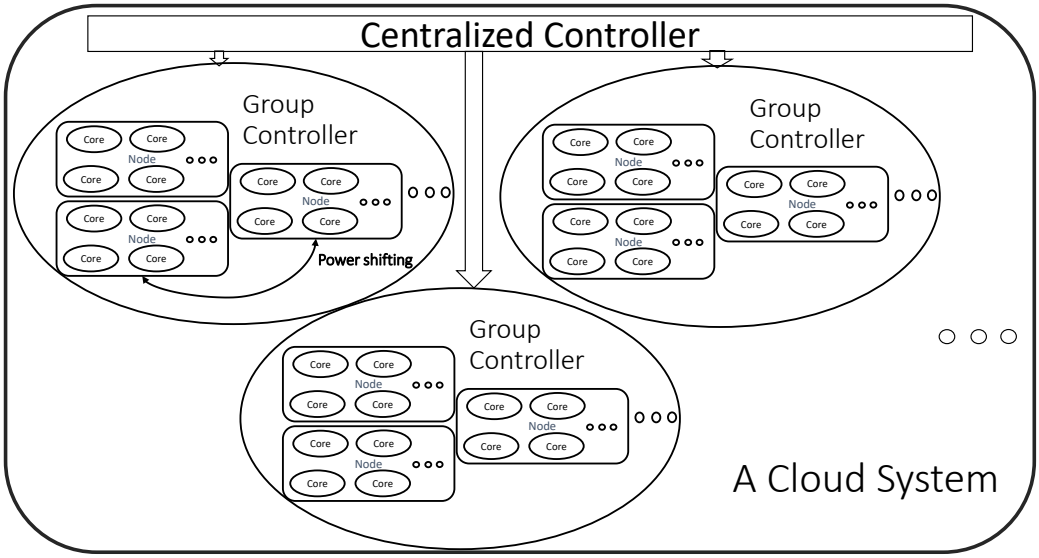

Figure 8.3: HPGPS Power Management

HPGPS works as follows:

- Grouping. The centralized controller divides nodes into groups, each group has fixed number of nodes. A group controller is assigned to each group.

- Power gating and shifting. Once a node reaches a barrier, the group controller power gates the node, and shifts the power budget from this node to other active nodes in the same group.

- Group size Computing. The group size is computed based on the power shifting communication delay, workload and the possibility to find a node in the power gating state. The power shifting communication delay means the time for nodes to communicate node status for power shifting, not the communication in the applications. The group controller compute group size is based on runtime information within the group and report that to the centralized controller. The new group size will be used for the next barrier.

We made three assumptions to simplify the HPGPS design:

Assumption 1 The group size is define ata the beginning of execution and remains unchanged until nodes are synchronized at a barrier. After that, the group size may be redefined by the centralized controller. In other words, group size can only be re-defined once the program execution reaches a barrier.

Assumption 2 Nodes at the same router should be put into the same group.

Assumption 3 Group size among all groups should be the same, except one group that may contain less nodes due to division remainder.

There are two main factors in making decision on group size.

- The power shifting communication delay within the group. Increasing group size leads to longer power shifting communication delay, while reducing group size will reduce power shifting communication time among nodes within the group.

- The probability to shift power among nodes. As presented in section 8.1, nodes are power gated after they arrive at the barrier. Power can be shifted only if the node is power gated. Hence, increasing group size will increasing the probability for power shifting since there are more nodes in the group.

Therefore, we need balance the trade-off between power shifting communication delay and power shifting probability. We model an on-line optimization problem to determine the group size based on runtime information, such as workload and network delay and use a stochastic approximation approach to solve the problem. The data are collected from real MPI traces and hardware measured power data is from a cloud system at AMD. Experiments are tested on an in house simulator at AMD [66].

In order to understand the impact of communication in HPGPS design, we tested HPGPS under 3 kinds of network delay and present the results in Figure 8.5. For a group with 100 nodes, the group delay is 0.0139ms, 0.9ms, and 5ms for those 3 cluster systems. For the same application, HPGPS computed group size for each system. We compare the performance speedup of HPGPS against the baseline execution and the power shifting and

power gating design. The baseline here is to execute the applications without power gating and power shifting techniques using the same computing resources. All results are normalized to the baseline. The speedup from power shifting and power gating design initially increases as system increases, but decreases after certain system size. When there are more nodes in the system, there is higher possibility to find nodes in power gate status. This increased power shifting contributes to the initial speedup increase. However, when the system size reaches some point, the power shifting communication delay among nodes is too large to compensate for the power shifting speedup. Those facts are illustrated in Figure 8.4. In fact, the power shifting communication delay can be even longer than the barrier execution. In that case, there is no chance to do any power shifting. However, HPGPS still achieves speedup in spite of large system size because of the hierarchical and grouping design. Nodes communicate for power shifting within the group instead of across the whole network.

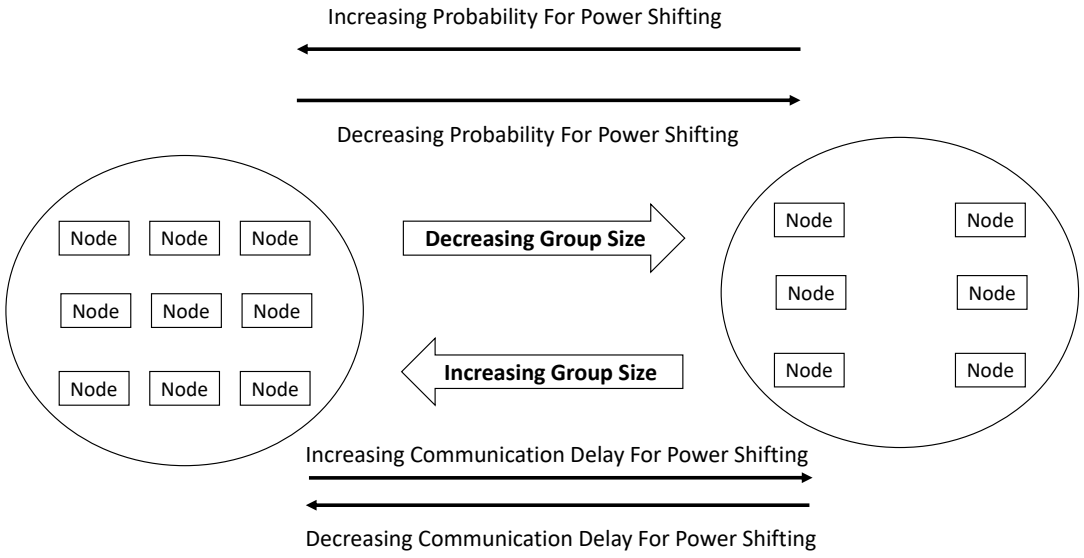

Figure 8.4: Probability vs Delay

Next we are going to use on-line optimization to determine the group size for HPGPS. Let's denote the group size for barrier i as ζ_i. The total number of nodes in the cloud system

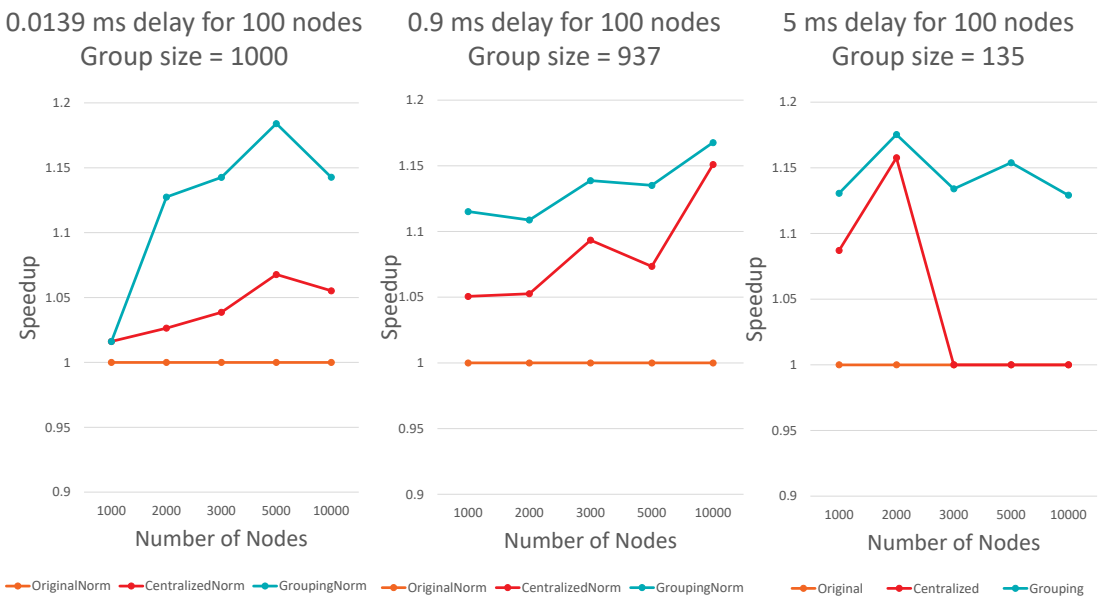

Figure 8.5: HPGPS Speedup vs Group Delay

is n. The power shifting communication delay within the group is defined as α_i, while the probability of one or more nodes complete execution during time interval α_i is defined as β_i, which is calculated from a distribution model obtained from regression analysis of the data at AMD. The node execution status are monitored on-line, with sampling interval α_i, which is the same as one time communication delay within the group. Let's define the time window α_i from the beginning of barrier i execution as k, $k = 1,2,3,\ldots$. Therefore, we formulate the following optimization model.

$$\max_{1 \le \zeta_i \le n} \frac{\beta_i}{\alpha_i} \tag{8.1}$$

We use online optimization algorithm to compute the optimal group size at each control cycle. For control cycle k, the group size is denoted as σ_k. Let's define

$$L_k = \frac{\beta_k}{\alpha_k} \tag{8.2}$$

So the group size at control cycle $k+1$ is computed by the following equation:

$$\sigma_{k+1} = \sigma_k + \theta_k * dL_k \tag{8.3}$$

where

$$\theta_k = \frac{1}{k^{0.6}} \tag{8.4}$$

8.2 Experimental Results

We tested HPGPS using traces for three applications AMR Miniapp, FillBoundary, and MultiGrid [91]. The AMR Miniapp is a compact proxy for octree-based AMR [92]. Fill Boundary is a proxy for evaluating communication patterns. MultiGrid is a proxy application for a solver. AMR Miniapp trace has 1728 nodes. Fill Boundary and Multigrid around 10,000. Figure 8.6, Figure 8.7, and Figure 8.8 shows the results for these three applications at different power budgets levels under 3 different network delays. The comparison baseline here is the original power shifting and power gating technique presented in [66]. Remember the maximum possible improvement, which reduce the barrier synchronization time to 0 is 12%. This is because our design migrates power only during synchronization in parallel program execution, not throughout the whole execution time. The portion of barrier synchronization in the tested programs is less than 12%. AMR and MultiGrid the centralized and the hierarchical power management mechanism delivers relative the same amount of performance. This is because for these applications there are just a small number of critical paths (the slowest processing threads in barrier synchronization). These nodes end up with extra power budget, but they cannot use it because the node frequencies reach the maximum point preventing further speedups.

The situation is different for FillBoundary, which has more critical paths (the slowest processing threads in barrier synchronization) and the distribution of the barrier arrival time

is more spread out. In this situation, HPGPS takes more time to react and shift the power reducing the opportunities for speedup. On the other hand, the hierarchical controller has more opportunities to shift the power within a group and this can be done much faster than the centralized, improving power shifting benefits and consequently performance.

There are two main advantages of HPGPS:

1. **Scalable property**. HPGPS depends on adaptive group size and can be applied to many could systems, regardless of system size. The original power shifting and power gating technique as [66] is not able to take action due to the communication delay. A single time node status communication throughout the supercomputer can be larger than the total barrier execution time when the system has larger number of nodes.

2. **Efficient Power Shifting** HPGPS shifts power more frequently than the original power gating and power shifting technique, leading to better performance and power efficiency in some applications. This advantage is due to the reduced communication delay within the group compared to the whole cloud system.

8.3 Concluding Remarks

This chapter presents HPGPS, a power management scheme to improve energy efficiency as well as performance for data centers. Compared to other related techniques [66], our design achieves up to 1.5% more energy saving. Furthermore, due to the hierarchical management, the design is also applicable to exascale cloud system.

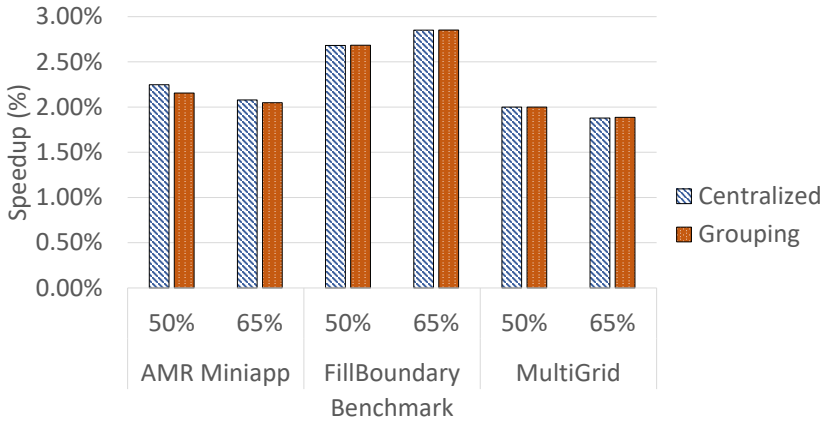

Figure 8.6: HPGPS Speedup vs Group Delay: Small Network Delay

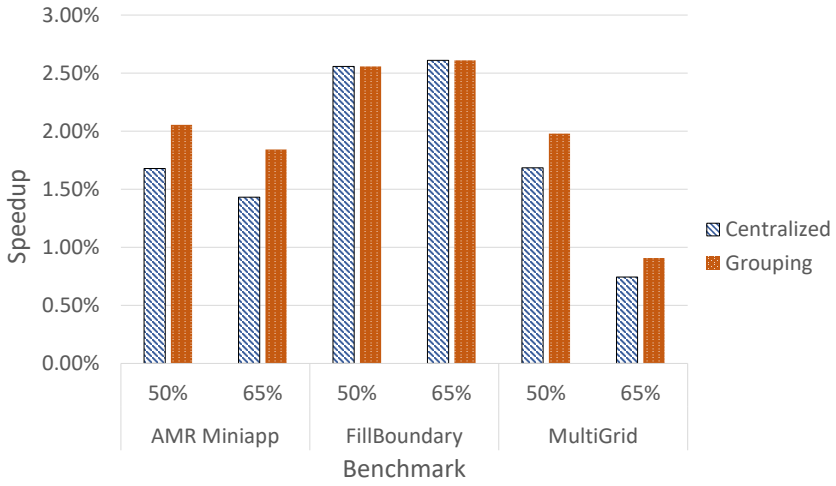

Figure 8.7: HPGPS Speedup vs Group Delay: Medium Network Delay

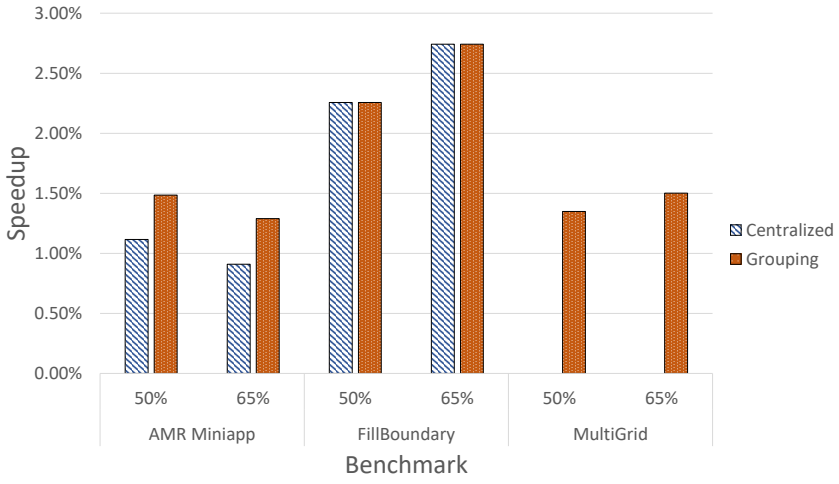

Figure 8.8: HPGPS Speedup vs Group Delay: Large Network Delay

CHAPTER 9
CONCLUSION

This dissertation addresses the problem of power and performance management for various computing systems, from single voltage island multicore processors to power-constrained extreme scale cloud systems. Balancing power and performance in modern computing systems is a complex optimization problem. This challenge is addressed by the statement of this thesis: Improving performance and power consumption in modern computing systems will require new techniques, and the body of control theories can provide the basis for such solutions. This thesis addresses this problem through three main contributions:

- Effective and efficient power & performance management techniques in a single voltage island multi-core processor.
- Maximizing power efficiency under a power cap in a multi-core processor that is composed of several voltage islands.
- A hierarchical power management technique to improve performance and energy efficiency under power budgets in a cloud system.

The processor level power and performance management is achieved by the design of adaptive gain feedback controllers. On the other hand, cloud level energy and performance management is obtained by hierarchical power-gating and power-shifting (HPGPS) using a stochastic approximation approach.

The first topic is comprised of 1) throughput regulation, 2) power regulation, and 3) power efficiency optimization, for single voltage island multicore processors. A throughput-frequency model is obtained by IPA analysis, while a power-frequency model is obtained by a system identification approach. Those models are generic that can be applied to various applications. They provide a foundation for the on-line optimization of power efficiency in

multicore processors. The power and throughput tracking controllers are general purpose regulators that are application independent, and the feedback gain is updated dynamically adapting to the workload.

The second topic addresses the problem of optimizing power efficiency in a many-core processor under power caps, such as those servers in the nodes of cloud computing systems. Given a power budget, we provide two techniques for improving the power efficiency: 1) an on-line optimization technique for maximizing throughput, 2) a dynamic power regulation technique that dynamically distributes power across the processor based on workload variation, which is an extension of the power regulation technique in the first topic. While those techniques have been studied individually, no significant efforts have been made to combine them, which could potentially be a future area of interest.

Finally the third topic addresses the problem of performance and energy efficiency improvement for cloud systems when there is a power cap. This work presents a hierarchical power gating & power shifting (HPGPS) technique for bulk synchronous parallel applications in cloud computing systems. Nodes that are otherwise waiting to be synchronized are power gated and their power budgets are redistributed to other high workload nodes, thus reducing the penalty of workload imbalances across the system. This technique is demonstrated to increase performance and decrease energy consumption in power constrained cloud systems. This hierarchical power management scheme is scalable to extreme scale cloud computing systems. A potential future research topic could be to combine the HPGPS with critical path predictions that can determine in advance which nodes will be power gated.

By examining these topics, this thesis provides power and performance management techniques for computing systems for single voltage island processors, multiple voltage islands servers, and cloud systems. The optimization techniques presented within this work help guide the power efficiency improvement of all kinds of computing systems.

This dissertation opens the door to future work in increasing power efficiency as well as

performance in various scales of computing systems, from portable devices to data centers. A future area of potential research could be combining the processor level optimization with HPGPS in data centers. The nodes can be implemented with the processor level optimization techniques proposed in this thesis, and the cloud system can be managed by HPGPS. Hopefully the work in this thesis helps to lay the foundation for these types of studies.

REFERENCES

[1] A Shehabi, S. Smith, N Horner, I Azevedo, R Brown, J Koomey, E Masanet, D Sartor, M Herrlin, and W Lintner, "United states data center energy usage report," *Lawrence Berkeley National Laboratory, Berkeley, California. LBNL-1005775 Page*, vol. 4, 2016.

[2] X. Chen, Y. Wardi, and S. Yalamanchili, "Instruction-throughput regulation in computer processors with data-center applications," *Discrete Event Dynamic Systems*, pp. 1–32, 2017.

[3] X Chen, Y Wardi, and S Yalamanchili, "Power regulation in high performance multicore processors," *arXiv preprint arXiv:1709.04859*, 2017.

[4] X. Chen, Y. Wardi, and S. Yalamanchili, "Ipa in the loop: Control design for throughput regulation in computer processors," in *Discrete Event Systems (WODES), 2016 13th International Workshop on*, IEEE, 2016, pp. 141–146.

[5] Y. Wardi, C. Seatzu, X. Chen, and S. Yalamanchili, "Performance regulation of event-driven dynamical systems using infinitesimal perturbation analysis," *Nonlinear Analysis: Hybrid Systems*, vol. 22, pp. 116–136, 2016.

[6] X Chen, H Xiao, Y. Wardi, and S. Yalamanchili, "Throughput regulation in shared memory multicore processors," in *High Performance Computing (HiPC), 2015 IEEE 22nd International Conference on*, IEEE, 2015, pp. 12–20.

[7] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture*, IEEE Computer Society, 2006, pp. 347–358.

[8] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, IEEE, 2008, pp. 363–374.

[9] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback control of computing systems*. John Wiley & Sons, 2004.

[10] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No power struggles: Coordinated multi-level power management for the data center," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 36, 2008, pp. 48–59.

[11] A. K. Mishra, S. Srikantaiah, M. Kandemir, and C. R. Das, "Cpm in cmps: Coordinated power management in chip-multiprocessors," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE Computer Society, 2010, pp. 1–12.

[12] N. Almoosa, W. Song, Y. Wardi, and S. Yalamanchili, "A power capping controller for multicore processors," in *American Control Conference (ACC), 2012*, IEEE, 2012, pp. 4709–4714.

[13] V. Krishnaswamy, J. Brooks, G. Konstadinidis, C. McAllister, H. Pham, S. Turullols, J. L. Shin, Y. YangGong, and H. Zhang, "4.3 fine-grained adaptive power management of the sparc m7 processor," in *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, IEEE, 2015, pp. 1–3.

[14] A. Deval, A. Ananthakrishnan, and C. Forbell, "Power management on 14 nm intel® core- m processor," in *Low-Power and High-Speed Chips (COOL CHIPS XVIII), 2015 IEEE Symposium in*, IEEE, 2015, pp. 1–3.

[15] X. Wang, K. Ma, and Y. Wang, "Adaptive power control with online model estimation for chip multiprocessors," *IEEE Transactions on parallel and Distributed Systems*, vol. 22, no. 10, pp. 1681–1696, 2011.

[16] Y. Yao and Z. Lu, "Fuzzy flow regulation for network-on-chip based chip multiprocessors systems," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, IEEE, 2014, pp. 343–348.

[17] G. M. Almeida, R. Busseuil, L. Ost, F. Bruguier, G. Sassatelli, P. Benoit, L. Torres, and M. Robert, "Pi and pid regulation approaches for performance-constrained adaptive multiprocessor system-on-chip," *IEEE Embedded Systems Letters*, vol. 3, no. 3, pp. 77–80, 2011.

[18] U. Brinkschulte and M. Pacher, "A theoretical examination of a self-adaptation approach to improve the real-time capabilities in multi-threaded microprocessors," in *Self-Adaptive and Self-Organizing Systems, 2009. SASO'09. Third IEEE International Conference on*, IEEE, 2009, pp. 136–143.

[19] N. Almoosa, W Song, S. Yalamanchili, and Y. Wardi, "Throughput regulation in multicore processors via ipa," in *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, IEEE, 2012, pp. 7267–7272.

[20] V. Hanumaiah and S. Vrudhula, "Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 349–360, 2014.

[21] X. Lin, Y. Xue, P. Bogdan, Y. Wang, S. Garg, and M. Pedram, "Power-aware virtual machine mapping in the data-center-on-a-chip paradigm," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*, IEEE, 2016, pp. 241–248.

[22] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1677–1690, 2011.

[23] M. Becchi and P. Crowley, "Dynamic thread assignment on heterogeneous multiprocessor architectures," in *Proceedings of the 3rd conference on Computing frontiers*, ACM, 2006, pp. 29–40.

[24] S. Kawaguchi and T. Yachi, "Adaptive power efficiency control by computer power consumption prediction using performance counters," *IEEE Transactions on Industry Applications*, vol. 52, no. 1, pp. 407–413, 2016.

[25] J. Meng, K. Kawakami, and A. K. Coskun, "Optimizing energy efficiency of 3-d multicore systems with stacked dram under power and thermal constraints," in *Proceedings of the 49th Annual Design Automation Conference*, ACM, 2012, pp. 648–655.

[26] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 9, pp. 1395–1408, 2010.

[27] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Voltage and frequency control with adaptive reaction time in multiple-clock-domain processors," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, IEEE, 2005, pp. 178–189.

[28] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, "Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, IEEE, 2006, 8–pp.

[29] B. Mochocki, D. Rajan, X. S. Hu, C. Poellabauer, K. Otten, and T. Chantem, "Network-aware dynamic voltage and frequency scaling," in *Real Time and Embedded Technology and Applications Symposium, 2007. RTAS'07. 13th IEEE*, IEEE, 2007, pp. 215–224.

[30] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs," in *SC 2006 conference, proceedings of the ACM/IEEE*, IEEE, 2006, pp. 14–14.

[31] Y.-H. Lu, L. Benini, and G. De Micheli, "Operating-system directed power reduction," in *Proceedings of the 2000 international symposium on Low power electronics and design*, ACM, 2000, pp. 37–42.

[32] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Ecosystem: Managing energy as a first class operating system resource," in *ACM SIGPLAN Notices*, ACM, vol. 37, 2002, pp. 123–132.

[33] H. Li, C.-Y. Cher, T. Vijaykumar, and K. Roy, "Vsv: L2-miss-driven variable supply-voltage scaling for low power," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, IEEE, 2003, pp. 19–28.

[34] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, 2006, pp. 359–370.

[35] R. Bianchini and R. Rajamony, "Power and energy management for server systems," *Computer*, vol. 37, no. 11, pp. 68–76, 2004.

[36] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, 2003, p. 93.

[37] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 35, 2007, pp. 13–23.

[38] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A comparison of high-level full-system power models.," *HotPower*, vol. 8, pp. 3–3, 2008.

[39] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the 2008 conference on Power aware computing and systems*, USENIX Association, 2008, pp. 10–10.

[40] S. Garg, D. Marculescu, and R. Marculescu, "Custom feedback control: Enabling truly scalable on-chip power management for mpsocs," in *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, IEEE, 2010, pp. 425–430.

[41] U. Y. Ogras, R. Marculescu, D. Marculescu, and E. G. Jung, "Design and management of voltage-frequency island partitioned networks-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 330–341, 2009.

[42] P. Bogdan, R. Marculescu, S. Jain, and R. T. Gavila, "An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platforms under highly variable workloads," in *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, IEEE, 2012, pp. 35–42.

[43] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark, "Formal control techniques for power-performance management," *IEEE micro*, vol. 25, no. 5, pp. 52–62, 2005.

[44] P. Bogdan and Y. Xue, "Mathematical models and control algorithms for dynamic optimization of multicore platforms: A complex dynamics approach," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, IEEE Press, 2015, pp. 170–175.

[45] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 170–183, 2013.

[46] S. Hemmert, "Green hpc: From nice to necessity," *Computing in Science & Engineering*, vol. 12, no. 6, pp. 8–10, 2010.

[47] D. Dharwar, S. S. Bhat, V. Srinivasan, D. Sarma, and P. K. Banerjee, "Approaches towards energy-efficiency in the cloud for emerging markets," in *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on*, IEEE, 2012, pp. 1–6.

[48] H. Yuan, C.-C. J. Kuo, and I. Ahmad, "Energy efficiency in data centers and cloud-based multimedia services: An overview and future directions," in *Green Computing Conference, 2010 International*, IEEE, 2010, pp. 375–382.

[49] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "A performance-conserving approach for reducing peak power consumption in server systems," in *Proceedings of the 19th annual international conference on Supercomputing*, ACM, 2005, pp. 293–302.

[50] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Optimizing job performance under a given power constraint in hpc centers," in *Green Computing Conference, 2010 International*, IEEE, 2010, pp. 257–267.

[51] B. Rountree, D. K. Lownenthal, B. R. De Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making dvs practical for complex hpc applications," in *Proceedings of the 23rd international conference on Supercomputing*, ACM, 2009, pp. 460–469.

[52] D. J. Kerbyson, A. Vishnu, and K. J. Barker, "Energy templates: Exploiting application information to save energy," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, IEEE, 2011, pp. 225–233.

[53] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Parallel job scheduling for power constrained hpc systems," *Parallel Computing*, vol. 38, no. 12, pp. 615–630, 2012.

[54] D. Li, B. R. De Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos, "Hybrid mpi/openmp power-aware computing," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, IEEE, 2010, pp. 1–12.

[55] C.-h. Hsu and W.-c. Feng, "A power-aware run-time system for high-performance computing," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, IEEE Computer Society, 2005, p. 1.

[56] V. W. Freeh, N. Kappiah, D. K. Lowenthal, and T. K. Bletsch, "Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs," *Journal of Parallel and Distributed Computing*, vol. 68, no. 9, pp. 1175–1185, 2008.

[57] C. Liu, A. Sivasubramaniam, M. Kandemir, and M. J. Irwin, "Exploiting barriers to optimize power consumption of cmps," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, IEEE, 2005, 10–pp.

[58] K. Kandalla, E. P. Mancini, S. Sur, and D. K. Panda, "Designing power-aware collective communication algorithms for infiniband clusters," in *Parallel Processing (ICPP), 2010 39th International Conference on*, IEEE, 2010, pp. 218–227.

[59] V. W. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer, "Exploring the energy-time tradeoff in mpi programs on a power-scalable cluster," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, IEEE, 2005, 10–pp.

[60] R. Ge, X. Feng, and K. W. Cameron, "Modeling and evaluating energy-performance efficiency of parallel processing on multicore based power aware systems," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, IEEE, 2009, pp. 1–8.

[61] K. Agarwal, K. Nowka, H. Deogun, and D. Sylvester, "Power gating with multiple sleep modes," in *Proceedings of the 7th International Symposium on Quality Electronic Design*, IEEE Computer Society, 2006, pp. 633–637.

[62] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, "Power management of datacenter workloads using per-core power gating," *IEEE Computer Architecture Letters*, vol. 8, no. 2, pp. 48–51, 2009.

[63] T. N. Miller, X. Pan, R. Thomas, N. Sedaghati, and R. Teodorescu, "Booster: Reactive core acceleration for mitigating the effects of process variation and application imbalance in low-voltage chips," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, IEEE, 2012, pp. 1–12.

[64] F. N. Tan, S. G. Pang, L. K. Yong, and C. S. Lee, "Power gating techniques on platform controller hub," in *Electronic Manufacturing Technology Symposium (IEMT), 2010 34th IEEE/CPMT International*, IEEE, 2010, pp. 1–7.

[65] L. Jian *et al.*, "Power shifting in thrifty interconnection network," in *Proc. High Performance Computer Architecture (HPCA)*, 2011.

[66] L. Piga, I. Paul, and W. Huang, "Performance boosting opportunities under communication imbalance in power-constrained hpc clusters," in *Parallel Processing (ICPP), 2016 45th International Conference on*, IEEE, 2016, pp. 31–40.

[67] J. Wang, J. Beu, R. Bheda, T. Conte, Z. Dong, C. Kersey, M. Rasquinha, G. Riley, W. Song, H. Xiao, *et al.*, "Manifold: A parallel simulation framework for multicore systems," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, IEEE, 2014, pp. 106–115.

[68] W. J. Song, S. Mukhopadhyay, and S. Yalamanchili, "Energy introspector: A parallel, composable framework for integrated power-reliability-thermal modeling for multicore architectures," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, IEEE, 2014, pp. 143–144.

[69] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, IEEE, 2009, pp. 469–480.

[70] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, "3d-ice: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling," in *Proceedings of the International Conference on Computer-Aided Design*, IEEE Press, 2010, pp. 463–470.

[71] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, *et al.*, "Haswell: The fourth-generation intel core processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, 2014.

[72] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring energy and power with papi," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, IEEE, 2012, pp. 262–268.

[73] (). Https://wiki.archlinux.org/index.php/cpu_frequency_scaling.

[74] (). Https://www.kernel.org/doc/documentation/cpu-freq/governors.txt.

[75] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*, IEEE, 1995, pp. 24–36.

[76] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ACM, 2008, pp. 72–81.

[77] L. Nai, Y. Xia, I. G. Tanase, H. Kim, and C.-Y. Lin, "Graphbig: Understanding graph computing in the context of industrial solutions," in *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*, IEEE, 2015, pp. 1–12.

[78] I. Tanase, Y. Xia, L. Nai, Y. Liu, W. Tan, J. Crawford, and C.-Y. Lin, "A highly efficient runtime and graph library for large scale graph analytics," in *Proceedings of Workshop on GRAph Data management Experiences and Systems*, ACM, 2014, pp. 1–6.

[79] P. Lancaster, "Error analysis for the newton-raphson method," *Numerische Mathematik*, vol. 9, no. 1, pp. 55–68, 1966.

[80] Y.-C. L. Ho and X.-R. Cao, *Perturbation analysis of discrete event dynamic systems*. Springer Science & Business Media, 2012, vol. 145.

[81] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[82] J. L. Hennessy and D. A. Patterson, *Computer architecture: A quantitative approach*. Elsevier, 2011.

[83] B. Goel and S. A. McKee, "A methodology for modeling dynamic and static power consumption for multicore processors," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*, IEEE, 2016, pp. 273–282.

[84] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. N. Strenski, and P. G. Emma, "Optimizing pipelines for power and performance," in *Microarchitecture, 2002.(MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*, IEEE, 2002, pp. 333–344.

[85] G. Sun, C. G. Cassandras, Y. Wardi, C. G. Panayiotou, and G. F. Riley, "Perturbation analysis and optimization of stochastic flow networks," *IEEE Transactions on Automatic Control*, vol. 49, no. 12, pp. 2143–2159, 2004.

[86] D. Clark and H. J. Kushner, *Stochastic approximation for constrained and unconstrained systems*, 1978.

[87] C. G. Cassandras, Y. Wardi, C. G. Panayiotou, and C. Yao, "Perturbation analysis and optimization of stochastic hybrid systems," *European Journal of Control*, vol. 16, no. 6, pp. 642–661, 2010.

[88] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed systems*, vol. 22, no. 6, pp. 931–945, 2011.

[89] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: Performance degradation due to nearby jobs," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ACM, 2013, p. 41.

[90] G. Shipman, P. McCormick, K. Pedretti, S. L. Olivier, K. B. Ferreira, R. Sankaran, S. Treichler, A. Aiken, and M. Bauer, "Analysis of application sensitivity to system performance variability in a dynamic task based runtime.," Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2016.

[91] U. DOE, "Characterization of the doe mini-apps," *Retrieved July*, vol. 14, 2016.

[92] A. Sasidharan and M. Snir, "Miniamr-a miniapp for adaptive mesh refinement," Tech. Rep., 2016.